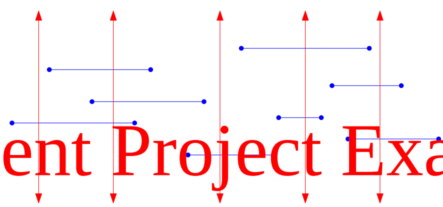
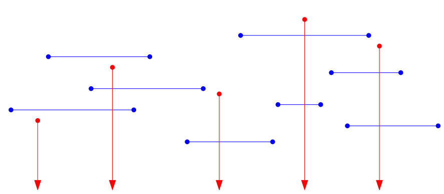
CS/ECE 374 A (Spring 2022)
Homework 5 Solutions

Problem 5.1:

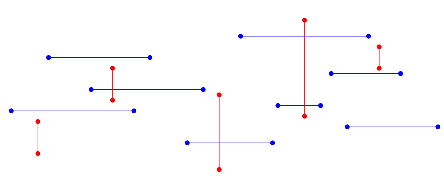
(a) (20 pts) Suppose that we are given a set H of horizontal line segments and a set V of vertical lines with $|H| + |V| = n$. (A horizontal line segment has two endpoints and can be specified by two x-coordinates and one y-coordinate; a vertical line is unbounded from above and from below, and can be specified by one x-coordinate.) Describe an $O(n \log n)$-time algorithm to count the total number of intersections between H and V. You may use sorting as a subroutine.



(b) (70 pts) Next, suppose that we are given a set H of horizontal line segments and a set V of vertical downward rays with $|H| + |V| = n$. (A vertical downward ray is unbounded from below, and can be specified by the x- and y-coordinates of its top endpoint.) Describe an algorithm to count the total number of intersections between H and V. Your algorithm should use divide-and-conquer and have running time $O(n \log^2 n)$ or better. You may (and should) use part (a) as a subroutine. [Hint: divide using a median horizontal line...]



(c) (10 pts) Finally, suppose that we are given a set H of horizontal line segments and a set V of vertical line segments with $|H| + |V| = n$. Describe an algorithm to count the total number of intersections between H and V. Your algorithm should have running time $O(n \log^3 n)$ or better. You should use part (b) as a subroutine. [Hint: one way is to use divide-and-conquer again, but there is also a slicker way, using (b)...]



[Note: You may assume that all x-coordinates and all y-coordinates are distinct.]

Solution:

(a) *Idea.* We sort all the x-coordinates of the endpoints of the horizontal segments *and* the vertical rays. We consider a vertical "sweep line" moving from left to right, and maintain a *depth* holding the number of horizontal segments intersecting the current sweep line. We increment/decrement *depth* whenever the sweep line passes through a left/right endpoint.

Pseudocode. The input is a set H of horizontal segments and V of vertical lines with $n = |H| + |V|$. We let *count* be the number of intersections found.

part-a(H, V):
1. sort the list X of the x-coordinates of all left and right endpoints of H
and the x-coordinates of all lines in V
2. $\mathit{count} = \mathit{depth} = 0$
3. for each $x \in X$ in increasing order do
4.   if x is the x-coordinate of a left endpoint in H then
5. $\mathit{depth} = \mathit{depth} + 1$
6.   else if x is the x-coordinate of a right endpoint in H then
7. $\mathit{depth} = \mathit{depth} - 1$
8.   else if x is the x-coordinate of a vertical line in V then
9. $\mathit{count} = \mathit{count} + \mathit{depth}$
10. return *count*

[Note: in actual implementation, each element of X would be a record containing an x-coordinate and its type (whether it is from a left or right endpoint of a horizontal segment, or from a vertical line); in case of a vertical line, the record would also contain a pointer to the line. This way, the conditions in lines 4, 6, and 8 can indeed be tested in constant time.]

Analysis. $|X| \leq 2n$ (each horizontal segment has two x-coordinates and each vertical line has one). Line 1 takes $O(n \log n)$ time by heapsort, for example. The linear scan in lines 2–10 takes $O(n)$ time. The total time is $O(n \log n)$.

[*Remark.* Alternatively, one could use binary search to compute the number of intersections for each horizontal line segment, after sorting all the vertical lines. This would also give $O(n \log n)$ total time.]

(b) *Idea.* We use a divide-and-conquer based on the median y-coordinate.

Pseudocode. The input is a set H of horizontal segments and V of vertical rays with $n = |H| + |V|$. We let *count* be the number of intersections found.

intersect-count(H, V):
1. if $n \leq 1$ then set count to 0 and return
2. let y_m be the median among all y-coordinates from both
the horizontal segments in H and the endpoints of the vertical rays in V
3. let H_U be the set of all horizontal segments above $y = y_m$
and H_L be the set of all horizontal segments below $y = y_m$

4. let V_U be the set of all vertical rays with endpoints above $y = y_m$ and V_L be the set of all vertical rays with endpoints below $y = y_m$
5. let V'_U be the set of lines obtained by extending the rays in V_U
6. return intersect-count(H_U, V_U) + intersect-count(H_L, V_L) + part-a(H_L, V'_U)

Explanation. The two recursive calls in line 6 handle intersections between H_U and V_U and intersections between H_L and V_L. There are no intersections betwen H_U and V_L, but we still have to consider intersections between H_L and V_U. The key observation is that below $y = y_m$, the rays in V_U are equivalent to lines; thus, the intersections between H_L and V_U can be found by calling the subroutine in part (a) for H_L and V'_U, as done in line 6.

Analysis. Let $T(n)$ be the running time for an input set of size n. Line 2 takes $O(n \log n)$ time by sorting the y-coordinates (or $O(n)$ time by a selection algorithm). Lines 3–5 clearly take $O(n)$ time. Line 6 takes $2T(n/2)$ time (ignoring floors and ceilings). We get the following recurrence:

$$T(n) = \begin{cases} 2T(n/2) + O(n \log n) & \text{if } n > 1 \\ O(1) & \text{if } n \le 1. \end{cases}$$

The recurrence is identical to one from Problem Old.5.2(b). For completeness, we re-describe one way to solve the recurrence, by iteration: for some constant c,

$$\begin{aligned} T(n) &\le 2T(n/2) + cn \log n \\ &\le 2[2T(n/4) + c(n/2)\log(n/2)] + cn \log n \;\le\; 4T(n/4) + 2cn \log n \\ &\le 4[2T(n/8) + c(n/4)\log(n/4)] + 2cn \log n \;\le\; 8T(n/8) + 3cn \log n \\ &\;\;\vdots \\ &\le 2^k T(n/2^k) + kcn \log n \\ &\le nT(1) + cn \log^2 n \qquad \text{by setting } k = \log n \\ &= O(n \log^2 n). \end{aligned}$$

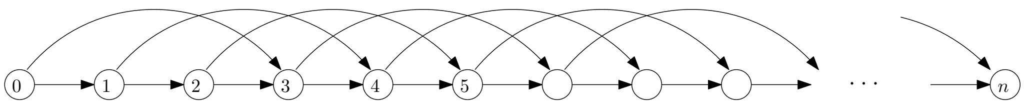
[*Remark.* The running time can be improved to $O(n \log n)$, for example, by pre-sorting the x- and y-coordinates once before the recursion starts. When the input is pre-sorted, the call to part-a in line 6 actually takes linear time, and given the sorted lists for (H, V), we can generate the sorted lists for (H_U, V_U) and (H_L, V_L) in lines 9–10 in linear time by a linear scan. So the recurrence becomes $T(n) = 2T(n/2) + O(n)$, which yields $O(n \log n)$ total time, even including the initial pre-sorting step.]

(c) The input is a set H of horizontal segments and V of vertical segments with $n = |H| + |V|$. For each vertical segment v in V, create a downward vertical ray v' that starts at the upper endpoint of v, and create another downward vertical ray v'' that starts at the lower endpoint of v. Observe that the number of intersections along v is exactly the number of intersections along v' minus the number of intersections along v''.
We call the subroutine from part (b) twice and subtract: the answer is precisely

$$\text{intersect-count}(H, \{v' : v \in V\} - \text{intersect-count}(H, \{v'' : v \in V\}).$$

We thus immediately obtain an $O(n \log^2 n)$-time algorithm for part (c) (or $O(n \log n)$-time according to the previous Remark).

Problem 5.2: Consider the following directed graph G_n:



We would like to count the number of the paths that go from vertex 0 to vertex n in G_n. Let X_n denote this number.

It is not difficult to see that X_n satisfies the recurrence

$$X_n \;=\; X_{n-1} + X_{n-3} \tag{1}$$

for all $n \geq 3$ (since we can enter vertex n either from vertex $n-1$ or from vertex $n-3$). For the base cases, $X_0 = X_1 = X_2 = 1$.

From this recurrence, it is straightforward to obtain an algorithm that computes X_n using $O(n)$ arithmetic operations. But we can do better. . .

(a) (10 pts) Prove that for all $m, n \geq 2$,

$$X_{m+n} \;=\; X_m X_n + X_{m-2} X_{n-1} + X_{m-1} X_{n-2}.$$

[Hint: in G_{m+n}, a path from vertex 0 to vertex $m+n$ may either go through vertex m, or skip over m in two possible ways. . .]

(b) (10 pts) Use part (a) (and Eq. (1))[1] to express $X_{2n}, X_{2n-1}, X_{2n-2}$ in terms of X_n, X_{n-1}, X_{n-2}.

(c) (45 pts) Using part (b), design and analyze an algorithm that computes X_n for a given n, using only $O(\log n)$ arithmetic operations.

(d) (35 pts) For large n, the number X_n is exponentially large (how many bits?), and so we can't assume that arithmetic operations take constant time. Show that your algorithm in part (c) can be implemented in $O(n^{\log_2 3})$ time, if multiplications are done using Karatsuba's algorithm. (In contrast, the naive approach using Eq. (1) would require $O(n^2)$ time when bit complexity is taken into account.)

[Note: If you are unable to do (a), you can still do (b)–(d), assuming the formula from (a).]

Solution:

(a) Any path from 0 to $m+n$ in G_{m+n} must satisfy *exactly* one of the following conditions:

- Type I: The path passes through m. The number of paths from 0 to m is X_m, and the number of paths from m to $m+n$ is X_n. Thus, the number of paths of this type is $X_m X_n$.

[1] It may be helpful to know, from Eq. (1), that $X_{n-3} = X_n - X_{n-1}$, for example.

- Type II: The path uses the edge from $m-2$ to $m+1$. The number of paths from 0 to $m-2$ is X_{m-2}, and the number of paths from $m+1$ to $m+n$ is X_{n-1}. Thus, the number of paths of this type is $X_{m-2}X_{n-1}$.
- Type III: The path uses the edge from $m-1$ to $m+2$. The number of paths from 0 to $m-1$ is X_{m-1}, and the number of paths from $m+2$ to $m+n$ is X_{n-2}. Thus, the number of paths of this type is $X_{m-1}X_{n-2}$.

It follows that the total number of paths from 0 to $m+n$ is given by $X_{m+n} = X_mX_n + X_{m-2}X_{n-1} + X_{m-1}X_{n-2}$.

(b)

$$
\begin{array}{lcll}
X_{2n} & = & X_n^2 + 2X_{n-1}X_{n-2} & \text{by (a) with } m = n \\
X_{2n-1} & = & X_{n-1}X_n + X_{n-3}X_{n-1} + X_{n-2}^2 & \text{by (a) with } m = n-1 \\
 & = & X_{n-1}X_n + (X_n - X_{n-1})X_{n-1} + X_{n-2}^2 & \text{since } X_{n-3} = X_n - X_{n-1} \\
 & = & 2X_nX_{n-1} - X_{n-1}^2 + X_{n-2}^2 & \\
X_{2n-2} & = & X_{n-1}^2 + 2X_{n-2}X_{n-3} & \text{by (a) with } m, n \text{ replaced by } n-1 \\
 & = & X_{n-1}^2 + 2(X_n - X_{n-1})X_{n-2} & \text{since } X_{n-3} = X_n - X_{n-1}
\end{array}
$$

(c) Given $n \geq 2$, the following algorithm returns the triple (X_n, X_{n-1}, X_{n-2}) (to compute X_n, we just call compute-triple(n) and extract the first argument of the output):

compute-triple(n):
1. if $n = 2$ then return $(1, 1, 1)$
2. if $n = 3$ then return $(2, 1, 1)$
2. $\quad (a, b, c) =$ compute-triple($\lfloor n/2 \rfloor$)
3. if n is even then
4. $\quad$ return $(a^2 + 2bc,\ 2ab - b^2 + c^2,\ b^2 + 2(a - b)c)$
5. else return $(a^2 + b^2 + 2ac,\ a^2 + 2bc,\ 2ab - b^2 + c^2)$

Explanation. Line 2 computes $a = X_{\lfloor n/2 \rfloor}$, $b = X_{\lfloor n/2 \rfloor - 1}$, and $c = X_{\lfloor n/2 \rfloor - 2}$. Correctness of the case of even n (line 4) follows directly from part (b) (with n replaced by $n/2$). For the case of odd n (line 5), part (b) (with n replaced by $\lfloor n/2 \rfloor$) gives $X_{n-1} = a^2 + 2bc$, $X_{n-2} = 2ab - b^2 + c^2$, and $X_{n-3} = b^2 + 2(a-b)c$. By Equation 1, we have $X_n = X_{n-1} + X_{n-3} = (a^2 + 2bc) + (b^2 + 2(a-b)c) = a^2 + b^2 + 2ac$.

Analysis. Let $T(n)$ be the number of arithmetic operations performed by compute-triple(n). Line 4 or 5 requires $O(1)$ arithmetic operations. Thus, we get the following recurrence:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + O(1) & \text{if } n > 2 \\ O(1) & \text{if } n \leq 2. \end{cases}$$

It is well known that this recurrence solves to $O(\log n)$ (it is the same recurrence as binary search).

(d) First note that $X_n = X_{n-1} + X_{n-3} \leq 2X_{n-1}$. It follows that $X_n \leq 2^n$, and so X_n is an $O(n)$-bit integer.

Redefine $T(n)$ to be the running time of compute-triple(n). Now, line 4 or 5 requires $O(1)$ additions/subtractions and multiplications of numbers with $O(n/2) = O(n)$ bits.

Each such addition/subtraction takes $O(n)$ time, and by Karatsuba's algorithm, each such multiplication takes $O(n^{\log_2 3})$ time. Thus, we get the following recurrence:

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + O(n^{\log_2 3}) & \text{if } n > 2 \\ O(1) & \text{if } n \leq 2. \end{cases}$$

One way to solve this recurrence is to apply the Master theorem (e.g., in Jeff's notes, Section II.3). Since $(n/2)^{\log_2 3} = \kappa n^{\log_2 3}$ with $\kappa = 1/3 < 1$, we have $T(n) = O(n^{1.59})$.

[Alternatively, we can directly expand the recurrence and obtain a geometric series: $T(n) \leq O(n^{\log_2 3} + (n/2)^{\log_2 3} + (n/4)^{\log_2 3} + \cdots) = O(n^{\log_2 3} \sum_{i=0}^{\infty} (1/2^i)^{\log_2 3}) = O(n^{\log_2 3} \sum_{i=0}^{\infty} (1/3)^i) = O(n^{\log_2 3})$.]

Alternate Solution to (a) by induction: (written by Daniel Christl)

Let $m \in \mathbb{N}$ with $m \geq 2$. Induction on n.

Base case: $n = 2, n = 3, n = 4$.
Then, we have $X_{m+n} = X_{m+2}$.
By the given recurrence,

$X_{m+2} = X_{m+1} + X_{m-1}$
$X_{m+2} = X_m + X_{m-1} + X_{m-2}$
Because $X_0 = X_1 = X_2 = 1$,
$X_{m+2} = X_m * X_2 + X_{m-1} * X_0 + X_{m-2} * X_1$

The same logic can be applied for cases 3, 4...

Inductive Hypothesis: Let $k \in \mathbb{N}$. with $k \geq 5$ and suppose
$X_{m+n} = X_m * X_n + X_{m-1} * X_{n-2} + X_{m-2} * X_{n-1}$ is true for all $2 \leq n \leq k$.

Inductive Step:
By the given recurrence,
$X_{m+(k+1)} = X_m * X_k + X_m * X_{k-2}$
By the Inductive Hypothesis,
$X_{m+(k+1)} = (X_m*X_k+X_{m-1}*X_{k-2}+X_{m-2}*X_{k-1})+(X_m*X_{k-2}+X_{m-1}*X_{k-4}+X_{m-2}*X_{k-3})$
Rearranging,
$X_{m+(k+1)} = X_m * (X_k + X_{k-2}) + X_{m-1} * (X_{k-2} + X_{k-4}) + X_{m-2} * (X_{k-1} + X_{k-3})$
By the given recurrence,
$X_{m+(k+1)} = X_m * X_{k+1} + X_{m-1} * X_{k-1} + X_{m-2} X_k$

$\therefore$ By the principle of strong induction, $X_{m+n} = X_m * X_n + X_{m-1} * X_{n-2} + X_{m-2} * X_{n-1}$ holds for all $n \in \mathbb{N}$ with $n \geq 2$.