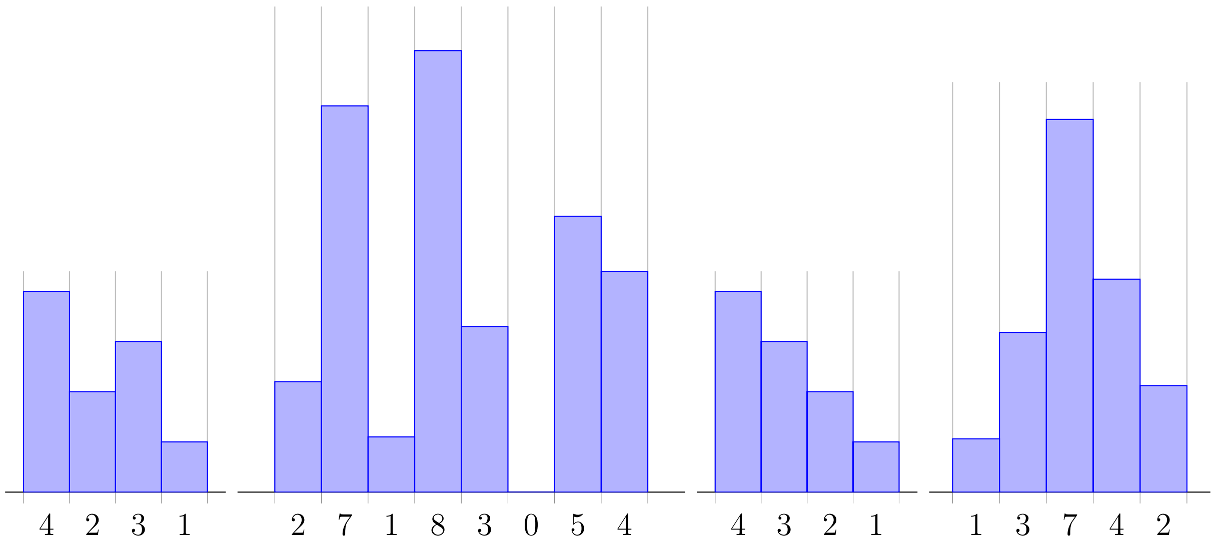
CS166 Problem Set 1: Range Minimum Queries

Luis A. Perez

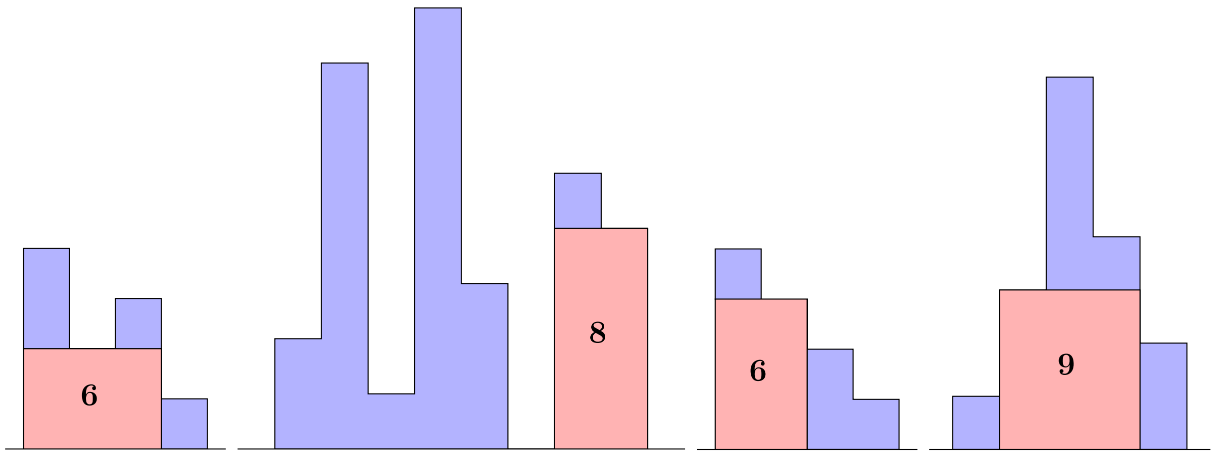
Problem One: Skylines (3 Points)

A ***skyline*** is a geometric figure consisting of a number of variable-height boxes of width 1 placed next to one another that all share the same baseline. Here's some example skylines, which might give you a better sense of where the name comes from:



Notice that a skyline can contain boxes of height 0. However, skylines can't contain boxes of negative height.

You're interested in finding the area of the largest axis-aligned rectangle that fits into a given skyline. For example, here are the largest rectangles you can fit into the above skylines:



Design an $O(n)$-time algorithm for this problem, where n is the number of constituent rectangles in the skyline. For simplicity, you can assume that no two boxes in the skyline have the same height. Follow the advice from our Problem Set Policies handout when writing up your solution – give a brief overview of how your algorithm works, describe it as clearly as possible, formally prove correctness, and then argue why the runtime is $O(n)$.

Solution: Overview

The key insight into this problem is that increasing the height or increasing the width of a box always lead to a box with higher area. Then we note that there exists at least one box of height equal to some rectangle (trivially, this is the box containing just that rectangle). As such, we can break the problem into (1) for each possible height, find the largest area of all boxes of that height and (2) out of those heights, find the largest area. At first this seems like a lot of work, but using RMQ, this process is actually contant time.

Next, we can be smart about how we approach this. For example, we consider the heights in increasing order. That is, find all boxes containing the smallest rectangle, and from those boxes, find the largest area. Trivially, the largest area in this case is simply the box spanning the entire range.

We now have the final insight. Any box of higher height *cannot* contain the minimum. This means that our minimum essentially splits the skyline into two parts, a left and a right. We can treat each of these as subproblems, and repeat the process. In this way, for each height, we will obtain the area of the largest box of that height. Then we simply return the maximum of these values.

Representation We assume the skyline is given simple as an array A of integers. From the problem statement, this array is of size n and contains only distinct values (no two skylines are the same height). Furthermore, $A[i]$ corresponds the the height of the i-th rectangle, from left-to-right, on the skyline.

Our algorithm will rely on using RMQs, and as such, we create an RMQ data structure which we call RMQ_A.

We will also keep an auxilary data structure, *largestAreaForBoxOfHeight* which is of size n, and where *largestAreaForBoxOfHeight*$[i]$ will store the area of the largest box for our skyline whose height is equal to the height of the i-th rectangle. For example, *largestAreaForBoxOfHeight*$[0]$ has the largest area of a box whose height is equal to that of the 0-th rectangle.

Algorithm Details We first construct RMQ_A. We note that, using the Fischer-Heun algorithm, this data structure can be constructed in $O(n)$ time. We further note that we can query it for the index of the minimum value in any range $[i, j]$ in $O(1)$ time.

We describe our algorith as working on a subarray, $A[i, \cdots, j]$. We can imagine it begins with $i = 0, j = n - 1$. For this subarray, first, we find the index of the minimum value contained in the subarray using our pre-computed RMQ_A. Let this index be k. In other words, $A[k] < A[l], \forall i \leq l \leq j$. We then compute $A[k] * (i - j + 1)$ and set this as the value for *largestAreaForBoxOfHeight*$[k]$.

We simply repeat the above process on the subarrays $A[i, \cdots, k-1]$ and $A[k+1, \cdots j]$ (as long as these subarrays contain at least one-element, otherwise we stop since there is no valid subarray).

In the end, the algorithm simply returns the maximum value found in *largestAreaForBox-OfHeight* as the answer.

Proof of Correctness

Analysis of Runtime The run-time analysis is rather straight-forward. Constructing the RMQ data-structure, as discussed in class, takes $O(n)$ time.

Next, we note that for each element that we add to *largestAreaForBoxOfHeight*, we perform a single RMQ query, which takes $O(1)$ times.

Next, we note that we never set a particular index of *largestAreaForBoxOfHeight* more than once, since once an index is set, the algorithm continues but on the two subarrays **not** containing that index. As such, we can repeat this process at most $O(n)$ times before the algorithm finishes.

Finally, we note that finding the maximum element in *largestAreaForBoxOfHeight* takes $O(n)$ time.

Putting it all together, our algorithm runes in $O(n) + O(n) * O(1) + O(n) = O(n)$, as desired.

Problem Two: Area Minimum Queries (4 Points)

In what follows, if A is a 2D array, we'll denote by $A[i, j]$ the entry at row i, column j, zero-indexed.

This problem concerns a two-dimensional variant of RMQ called the ***area minimum query*** problem, or ***AMQ***. In AMQ, you are given a fixed, two-dimensional array of values and will have some amount of time to preprocess that array. You'll then be asked to answer queries of the form "what is the smallest number contained in the rectangular region with upper-left corner (i, j) and lower-right corner (k, l)?" Mathematically, we'll define $AMQ_A((i, j), (k, l))$ to be $\min_{i \le s \le k, j \le t \le l} A[s, t]$. For example, consider the following array:

31	41	59	26	53	58	97
93	23	84	64	33	83	27
95	2	88	41	97	16	93
99	37	51	5	82	9	74
94	45	92	30	78	16	40
62	86	20	89	98	62	80

Here, $A[0, 0]$ is the upper-left corner, and $A[5, 6]$ is the lower-right corner. In this setting:

- $AMQ_A((0, 0), (5, 6)) = 2$
- $AMQ_A((0, 0), (0, 6)) = 26$
- $AMQ_A((2, 2), (3, 3)) = 5$

For the purposes of this problem, let m denote the number of rows in A and n the number of columns.

i. Design and describe an $\langle O(mn), O(\min\{m, n\})\rangle$-time data structure for AMQ.

Solution: Your solution goes here!

ii. Design and describe an $\langle O(mn \log m \log n), O(1)\rangle$-time data structure for AMQ.

Solution: Your solution goes here!

It turns out that you can improve these bounds all the way down to $\langle O(mn), O(1)\rangle$ using some very clever techniques. This might make for a fun final project topic if you've liked our discussion of RMQ so far!

Problem Three: Hybrid RMQ Structures (4 Points)

Let's begin with some new notation. For any $k \geq 0$, let's define the function $\mathbf{log^{(k)} n}$ to be the function:

$$\overbrace{\log \log \log \dots \log}^{k \text{ times}} n$$

For example:

$$\log^{(0)} n = n \qquad \log^{(1)} n = \log n \qquad \log^{(2)} n = \log \log n \qquad \log^{(3)} n = \log \log \log n$$

This question explores these sorts of repeated logarithms in the context of range minimum queries.

i. Using the hybrid framework, show that that for any fixed $k \geq 1$, there is an RMQ data structure with time complexity $\left\langle O\left(n \log^{(k)} n\right), O(1)\right\rangle$. For notational simplicity, we'll refer to the kth of these structures as D_k.

   Solution: Your solution goes here!

   (Yes, we know that the Fischer-Heun structure is a $\langle O(n), O(1) \rangle$ solution to RMQ and therefore technically meets these requirements. But for the purposes of this question, let's imagine that you didn't know that such a structure existed and were instead curious to see how fast an RMQ structure you could make without resorting to the Method of Four Russians. ☺)

ii. Although every D_k data structure has query time $O(1)$, the query times on the D_k structures will increase as k increases. Explain why this is the case and why this doesn't contradict your result from part (i).

   Solution: Your solution goes here!

Problem Four: Implementing RMQ Structures (10^+ Points)

This one is all coding, so you don't need to write anything here. Make sure to submit your final implementations on myth.