Version 1.1 ***due:*** *Mon, Sep 16 @ 11:59pm*

1 Introduction

This assignment is meant to help you practice implementing, analyzing, and proving the correctness of a divide and conquer algorithm, and to familiarize you with the sequence `scan` operation. In this assignment, you will code a solution to the Pittsburgh Skyline problem, which will help you learn all of those stated goals. We will go over `scan` in more detail this week in lecture.

You will likely find this assignment more conceptually challenging than the first assignment. We recommend starting early so you have time to ask for help.

2 Files

After downloading the assignment tarball from Autolab, extract the files from it by running `tar -xvf skylinelab-handout.tgz` from unix.andrew.cmu.edu. You should see the following files:

1. `skylinelab.pdf`
2. `bobbfile.py`
3. `Makefile`
4. `sources.cm`
5. `support.cm`
6. `lib/`
7. `SKYLINE.sig`
8. `Tester.sml`
9. `MkReferenceSkyline.sml`
10. * `MkSkyline.sml`
11. * `Tests.sml`

You should only modify the last 2 files, denoted by *. Additionally, you should create a file called `written.pdf` which contains the answers to the written part of the assignment.

Important note: You should extract the handout to the unix.andrew.cmu.edu clusters directly, and work within the directory created by extracting. This is because extracting on other computers can result in incorrect permission and lost hidden files.

3 Submission

There are two ways to submit your solutions. Before submitting, ensure that `written.pdf` exists and is a valid PDF document.

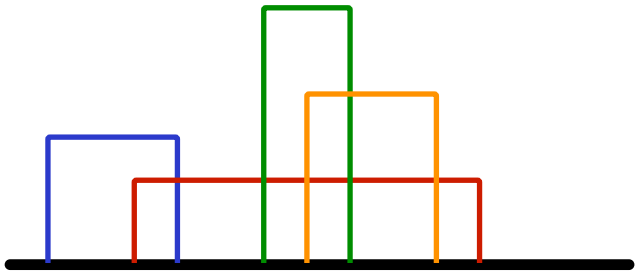
The first, and easiest way to submit is by running `./submit` from within the `skylinelab` directory. This will automatically package your submission into a tgz archive, and submit it to Autolab.

Important note: If you submit this way, you will still need to visit the Autolab website to view the results of the tests that Autolab runs against your code (which will contribute to part of your grade for this assignment). It is important to always do this for every lab to make sure that your code runs as you expect it to.

If you would rather submit through the Autolab website, run `./submit package` from the `skylinelab` directory. This should produce a file called `handin.tgz`. Open the Autolab webpage and submit this file via the "Handin your work" link.

4 Pittsburgh Skyline

Did you know the Pittsburgh skyline was rated the second most beautiful vista in America by USA Weekend in 2003? However, you won't get to go out and see it any time soon, because you're an overworked Carnegie Mellon student. The 15-210 staff felt sorry for you, so we recorded the location and the silhouette of each building in Pittsburgh. Using this data, you can calculate the silhouette of Pittsburgh's skyline.



In this instance of the problem, we will assume that each building silhouette b is a two-dimensional rectangle, represented as the triple (ℓ, h, r). The corners of the rectangle will be at $(\ell, 0)$, (ℓ, h), (r, h), and $(r, 0)$. That is, the base of the building runs along the ground from $x = \ell$ to $x = r$, and the building is h tall. (Contrary to popular belief, the city's ground is flat)

Definition 4.1 (The Pittsburgh Skyline Problem)**.** Given a non-empty set of buildings $B = \{b_1, b_2, \ldots, b_n\}$, where each $b_i = (\ell_i, h_i, r_i)$, the *Pittsburgh skyline problem* is to find a set of points $S = \{p_1, p_2, ..., p_{2n}\}$, where each $p_j = (x_j, y_j)$ such that

$$S = \left\{ \Big(x, \max(h : (\ell, h, r) \in B \wedge x \in [\ell, r)) \Big) : x \in \bigcup_{(\ell,h,r)\in B} \{\ell, r\} \right\}$$

In other words, the x-coordinates expressed in S are exactly the x-coordinates expressed in B, and the y-coordinate for every element $(x_j, y_j) \in S$ is the height of the tallest building $b_i = (\ell_i, h_i, r_i)$ for which $\ell_i \leq x_j < r_i$. The input set of buildings is represented as an unsorted sequence of tuples, and the output set is represented as a sequence of points sorted by x-coordinate.

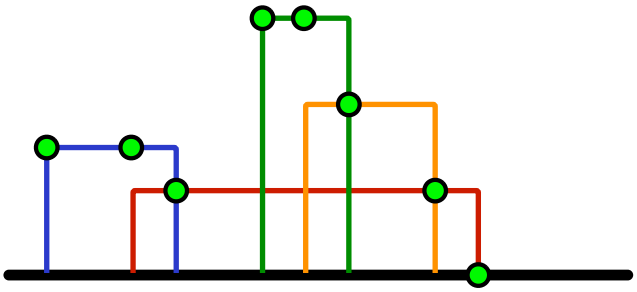
Input sequence:

ℓ_l	h_i	r_i
1	3	4
3	2	11
6	6	8
7	4	10

Output sequence:

x_j	y_j
1	3
3	3
4	2
6	6
7	6
8	4
10	2
11	0

Output points shown on input buildings:



4.1 Logistics

For this problem you will hand in file `MkSkyline.sml`, which should contain a functor ascribing to the signature `SKYLINE` defined in `SKYLINE.sml`. Your functor will take as a parameter a structure ascribing to the signature `SEQUENCE`, and implement the function `skyline : (int * int * int) seq -> (int * int) seq`.

4.2 Implementation Instructions

There are many algorithmic approaches to this problem, including brute force, sweep line, and divide and conquer. As you may have guessed, you will implement a work-optimal, low-span, divide-and-conquer solution to this problem. Even with divide and conquer, there are several choices on how to divide the problem. For example, you can divide along the x-axis, finding the skylines for $x < x'$ and for $x \geq x'$, or you can divide along the y-axis, finding the skylines for tall buildings and for short buildings. Alternately, you can divide the sequence of buildings into two sequences with a nearly equal number of buildings. You will use this last approach because it is easier to balance the recursive computations.

To simplify your solution, you may assume that all x-coordinates given in the input are unique, and that all heights and x-coordinates are non-negative integers.

You may have noticed that the definition of a skyline output sequence includes redundant points, and that removal of points with the same y-coordinate as the previous point would still result in a fully defined skyline. For example, the two points (3,3) and (7, 6) in the example above are redundant. **Omit these points from the sequence returned by your `skyline` function.**

The work of your divide-and-conquer steps must satisfy the recurrence

$$W_{skyline}(n) = 2W_{skyline}(n/2) + O(1) + W_{combine}(n), \tag{1}$$

where $W_{combine}(n)$ denotes the work for the combine step.

Task 4.1 (50%).

For full credit, **we expect your solution to have** $O(n \log n)$ **work and** $O(\log^2 n)$ **span**, where n is the number of input buildings.

The `copy_scan`, which will be discussed next week in lecture, may be helpful in your combine step.

There will be public tests, worth a total of 10% of your grade, and this score will appear on Autolab when you hand in your work. Some of these tests are given in `Tests.sml`). We will also run private tests when we grade, which will contribute to 10% of your grade. Algorithmic correctness and efficiency will be worth 30% and style will be worth 10%.

Note: You may receive **zero credit** for task 4.1 if you do any of the following:

- Hard code to the public tests.
- Submit a sequential solution.
- Submit the reference solution or a similar solution as your own.

Task 4.2 (10%). To aid with testing we have provided a testing structure, `Tester`, which should simplify the testing process. `Tester` will look at the file `Tests.sml`, in which you should put your test input.

At submission time there must not be any testing code in the same file as your `SKYLINE` implementation, as it can make it difficult for us to test your code. Additionally, unlike the 15-150 testing methodology, our testing structure does not test your code at compile time. In order to test your code, after running `CM.make`, you will need to run `Tester.testSkyline()` to test your implementation.

4.3 Proofs

In 15-150, you were encouraged to write correctness proofs by stepping through your code. *Do not do this in 15-210*. For complicated programs, that level of detail quickly becomes tedious for you and your TA. A helpful guideline is that you should prove that your algorithm is correct, not your code. But your proof should highlight the critical steps and describe your algorithm in sufficient detail that your algorithm maps straightforwardly onto your code. Consider Theorem 1.2 of Lecture 4, as an example of the level of detail we expect.

Task 4.3 (30%).

Prove the correctness of your divide-and-conquer algorithm by induction. Be sure to carefully state the theorem that you're proving and to note all the algorithm steps in your proof.

You should use the following structural induction principle for abstract sequences:

> Let P be a predicate on sequences. To prove that P holds for every sequence, it suffices to show the following:
>
> 1. $P(\langle\,\rangle)$ holds,
> 2. For all x, $P(\langle x \rangle)$ holds, and

3. For all sequences S_1 and S_2, if $P(S_1)$ and $P(S_2)$ hold, then $P(S_1@S_2)$ holds.

Task 4.4 (10%).

Carefully explain why your divide-and-conquer steps satisfy the specified recurrence and prove a closed-form solution to the recurrence.