CSC373H1S 2014 **Assignment A1**
Due: Thursday, January 30, 10PM Total Score: 10 points

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use *any* sources (textbooks, online notes, friends) please cite them for your own safety.

You can use those data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's running time is $O(n \lg n)$.

Questions

1. **Show Dog! (Bad Dog! Smelly Dog!)**

   Dogs – they're great. You love your dog so much that if you were ever separated, you would simply fall apart. Since your dog is simply the best dog, you have decided to enter him in the yearly UltraDog Special (the most prestigious of all dog shows, featuring the top dogs from every corner of the globe). Since the meet is only n days away, you have decided to put your dog on a rigourous training regimen. However, you don't want to overtrain your dog — a tired dog is not a good dog. Especially not a good show dog.

   So, starting from now (Day 1) until the UDS, you can choose each day to either train your dog or let your dog rest. However, the ultimate benefit that your dog receives from either training or taking a rest will depend *only* on the day (dogs live in the moment, after all). Formally, on the ith day ($1 \leq i \leq n$), training your dog will give it a benefit of t_i, while letting your dog rest will give a benefit of r_i. Resting takes only a day (leaving your dog ready and able to train – or rest – again tomorrow). On the other hand, the extra rigourous training regimen you learned from your parents (who tragically lost the UDS by a sliver exactly 50 years ago) is effective, but takes two days. You want to find a training regimen that will maximize your dogs benefit (and thus finally resolve the dying wish of your father).

   Therefore, define a *training regimen* to be a function $f : \{1, 2, \ldots, n\} \to \{T, R, *\}$, where for each i, $f(i) = T$ if you decide to train your dog on day i, $f(i) = *$ if you are continuing the training from day $i-1$, and $f(i) = R$ if you decide to let your dog rest on day i. The *benefit* of a training regimen is sum of the benefits of the training days and the rest days:

$$B = \sum_{i:f(i)=T} t_i + \sum_{i:f(i)=R} r_i.$$

   Input: Two lists of integers (positive or negative) $T = \{t_1, t_2, \ldots, t_{n-1}\}$, $R = \{r_1, r_2 \ldots, r_n\}$, where t_i represents the benefit to your dog if you train on day i, while r_i represents the benefit to your dog if you rest on day i.

   Output: The training regimen f that maximizes your dog's benefit B over the next n days, given by a choice of resting or training on each day.

   Construct an efficient algorithm that solves the above problem. Give a high-level description of how your algorithm works, state the running time of your algorithm and prove that it is optimal.

2. **Frugal Flying**

   You are a helicopter pilot for the FIRST-ACE helicopter company, and you have just been assigned to a new contract. The details of the contract are kept hidden (for super-secret security purposes), but you know this: you have been given a particular road to monitor (the road has length ℓ, for some positive integer ℓ). Every hour you will be sent a list of *non-intersecting* intervals on the road that you need to monitor (that is, a list of

non-decreasing integer pairs $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, where $I_j = [s_{I_j}, f_{I_j}] \in \mathbb{Z}^2$, with $s_I < f_I \leq \ell$ for all I) with a subtly-installed spy-cam on the bottom of your helicopter. Unfortunately, due to budget cuts, your spy-cam only has a limited amount of film, and once it turns on it can never be turned off. In particular, it can only film a stretch of the road of length $f \in \mathbb{Z}$. You can, however, choose when to turn it on – for example, if you choose to turn on the spy-cam at point x on the road, then it will film continuously until the point $x + (f - 1)$, after which it will turn off. When the spy-cam turns off you have to return to the start to re-load your film. Before you start to fly, then, your job is to figure out "start points" for filming on the road so that you can cover all of the input road-intervals $\mathcal{I}$ while minimizing the number of flights used.

Input: A positive integer $\ell \in \mathbb{Z}$ representing the length of the road, a positive integer f representing the length of film your spy-camera takes, and a list of intervals $\mathcal{I}$ where for all $I = [s_I, f_I] \in \mathcal{I}$ we have $s_I < f_I \leq \ell$.

Output: A sequence of filming points $c_1, c_2, \ldots, c_m \in \mathbb{Z}$, so that all of the intervals in $\mathcal{I}$ are covered by (possibly a union) of the intervals in $\{[c_1, c_1 + f), [c_2, c_2 + f), \ldots, [c_m, c_m + f)\}$ and m is minimized.

Construct an efficient algorithm which solves the above problem. Give a high-level description of how your algorithm works, state the time complexity of your algorithm, and prove that your algorithm is optimal.

3. **Successful Circuit Soldering**

You are the head of research for the largest circuit board company in the world — THUNDERCIRCUITS. Each circuit board created by THUNDERCIRCUITS is a rectangle, defined by a width W, a length L, and n circuit components $C_1, C_2, \ldots, C_n$. Furthermore, each circuit component C_i is another rectangle, defined by its own width w_i, length ℓ_i, and position of the bottom-left corner (x_i, y_i) relative to the bottom-left corner of the circuit board (which is $(0, 0)$). Each circuit component needs to be powered, and for this purpose a power component is attached to the bottom-left corner of every board (that is, at the point $(0, 0)$). A circuit component C_i can be powered by laying down a strip of solder connecting any of the corners of the circuit component to either the power source or a corner of another circuit component C_j which is already powered. For simplicity, assume that solder can only be put down in a straight line connecting two points. Your job is to come up with an efficient algorithm which, given a circuit board C and all of its components, outputs the minimum amount of solder needed to power all of the components.

Input: A circuit board C with its length L and width W, both positive real numbers. A list of circuit components $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, where each component C_i is defined by a triple $(w_i, \ell_i, (x_i, y_i))$ containing the width, length, and position of the bottom left corner of the component on the original circuit board, and note that all numbers defining a component are positive reals.

Output: The minimum amount of solder needed to power all of the components on the circuit board.

Construct an efficient algorithm for the above problem. Give a high-level description of how your algorithm works, state the time complexity of your algorithm, and prove that your algorithm is optimal.

4. **Scrabble®Time**

In the board game Scrabble — that is, the crossword game published by Hasbro since 1938 and loved by all — each character in the English language is assigned a point score based roughly on the "rarity" of that letter in day-to day speech. Letters which appear less often in words (such as Q and Z) have larger point values while common letters (such as E) have very low point values.

Here are the Scrabble point scores for the first ten letters of the alphabet:

Letter	Point Score
a	1
b	3
c	3
d	2
e	1
f	2
g	2
h	4
i	1
j	8

We will use these point scores to produce an efficient Huffman tree encoding of the first ten letters of the alphabet. Using these scores, define the *frequency weight* of a letter to be the inverse of its score – that is, if a letter x has a score $s(x)$, then the frequency weight of x will be $f(x) = 1/s(x)$.

Normalize the frequency weights of each letter to produce a probability distribution on the first ten letters of the alphabet, and then use this probability distribution to produce a Huffman tree. Is there another Huffman tree encoding for this set of probabilities? If so, show it, and if not, describe why not. Is there another Huffman tree encoding for this set of probabilities that is *not* just a relabeling of the leaves? (Again, if so, show it, and if not, describe why not).