Homework 3: Greedy

Be sure to read the grading guidelines and style guidelines. Especially to see the suggested format for describing algorithms.

We sometimes describe how long are justifications or proofs are. These lengths are intended to help you estimate how much detail we're expecting, you should not take those estimates as hard length-limitations.

Our solutions for any individual problem will fit in approximately one page or less. **Beginning with this homework, if you submit an answer substantially longer than 2 pages, the TAs are allowed to stop reading at the end of page 2.**

You are allowed (and encouraged!!) to collaborate with each other. Brainstorming is much easier to do in a group than alone! But you must follow the collaboration policy (which includes needing to write your submission on your own).

You will submit to gradescope; we will have a different box for each problem, so please give yourself extra time to submit.

1. Reduction! [10 points]

For a weighted graph $G = (V, E)$ (where the weights can be any real numbers), we call a set $S \subseteq E$ a "maximum spanning tree" if it is a spanning tree, and the sum of its edges is at least the sum of any other spanning tree.

You are given the following library function.

`MinSpanningTree`
Input: A weighted graph made up of a set of edges E and a set of vertices V where the i^{th} edge has weight w_i
Output: A minimum-weight set of edges such that you can get from any vertex of G to any other on only those edges

Explain how to use this library function to find the maximum spanning tree of a graph G, and briefly explain why it works (our full answer is only 3-4 setences).

2. Laksurk's Algorithm [25 points]

Instead of adding edges, we propose a new algorithm for solving the Minimum Spanning Tree problem by removing edges:

Let $G = (V, E)$ be a graph where edge i has weight w_i. Call the edge e "unnecessary" if removing e would not cause the graph to be disconnected.

Consider the following greedy algorithm

function `Laksurk's-Algorithm`(Graph G)
  Sort edges in **decreasing** order of edge weight.
  for each edge e in decreasing order **do**
    if e is unnecessary **then**
      delete e
  return remaining edges, which are an MST.

(a) Prove that if G is a connected graph, then `Laksurk's Algorithm` produces a spanning tree (don't worry about minimum yet!) [5 points]

(b) Prove that if G is a connected graph with distinct edge weights, then `Laksurk's Algorithm` produces the minimum spanning tree.
You may use as a fact that since the edge weights are distinct, the MST is unique. [20 points]
Hint: We strongly recommend using a structural result here. You should think about the structural result from class and see what the corresponding one here might be.

3. Reasonable Minds May Differ [25 points]

Sometimes, a seemingly reasonable greedy idea might not work. Your friend is trying to solve the "Positive Interval Covering" problem:

In an array A of integers, call an interval i, j **positive** if $A[i] + A[i+1] + \cdots + A[j] > 0$. Your friend wishes to "cover" the positive integers of the array with **disjoint** intervals (i.e., the intervals may not overlap). An element is "covered" if it is part of one of the intervals. Given an array A, containing n integers (which may be positive, negative, or zero), your friend wants to find the smallest number of disjoint positive intervals which can cover all the positive elements of A. Note that negative elements (and copies of 0) may be uncovered, or they may be part of intervals.

Your greedy friend loves using greedy algorithms, and has a greedy idea to solve this problem. Their idea is: starting from the leftmost positive element, keep a running sum of the elements in the interval as you move the right endpoint to the right. Move it to the right until the interval becomes non-positive, then make everything to the left the interval.

Your friend's idea seems quite reasonable on the surface. Being the great friend you are, you've decided to help your friend write some pseudocode for their idea. Of course, being the amazing coder that you are, you've perfectly implemented your friend's idea in the following perfectly perfect pseudocode:

```
1:  function PositiveCovering(A[1..n])
2:      startInd ← 1
3:      while There is a positive element in A at startInd or later do
4:          while A[startInd] ≤ 0 do                ▷ Find first uncovered positive element
5:              startInd++
6:                                                  ▷ move right until invalid
7:          endInd ← startInd+1
8:          sum ← A[startInd]
9:          while sum > 0 do
10:             if endInd = n + 1 then   ▷ [startInd, endInd-1] is a valid interval, and we're at the end of the array
11:                 endInd++                        ▷ When we exit the loop, we want to include element n
12:                 break
13:             sum+ = A[endInd]
14:             endInd++
15:         Add [startInd, endInd − 1) to list of intervals   ▷ Only include elements when sum was positive
16:         startInd ← endInd − 1                   ▷ first uncovered element remaining
        return list of intervals
```

Your friend knows that every greedy idea needs a proof of correctness, and they wrote the following attempt at a proof of correctness:

Spoof. We prove the claim using the "greedy stays ahead" format.

Let OPT be an optimal solution, and let ALG be the solution generated by our algorithm. We will show that for all n, after the n^{th} interval (reading from left-to-right) ALG has covered at least as many positive elements as OPT.

We proceed by induction on n.

Base Case: $n = 1$
The left-most interval must cover the left-most positive element. ALG chooses the longest positive interval that includes that first element (as it runs until the interval becomes invalid), so it must have at least as many elements covered as OPT does.

IH: Suppose that the claim holds for $n = 1, ..., k$.
IS: Let $i...j$ be the endpoints of the $(k+1)^{\text{st}}$ interval (from left-to-right) of OPT. We claim that ALG will cover at least through index j.

Case 1: ALG was testing starting from i (that is, index i contained the left-most uncovered positive element)
Since OPT takes $i, ..., j$ as an interval, it must be a positive interval. Thus ALG will consider $i...j$ as a valid interval.

It will only choose a different interval if it is longer, since we only ever increase `endInd`. Thus ALG covers at least as many elements as OPT.

Case 2: ALG was testing starting from an index $\ell \neq i$
By IH, ALG covered at least as many elements as OPT in the first k intervals. Thus we have $\ell > i$. Observe that any elements at or after i but strictly before ℓ (i.e. those that OPT covers but ALG does not) will not include any positive entries, as ALG starts at the first uncovered positive element. Thus, the sum $A[i] + \cdots + A[j]$ is at most $A[\ell] + \cdots + A[j]$. Since OPT considers $i...j$ a positive interval, $\ell, ..., j$ is also a positive interval, which means that ALG will consider that interval. Thus ALG also will choose an ending point j or larger.

In both cases, we have shown the claim holds for $k + 1$, as required. □

(a) One of your TAs notices your pseudocode and tells you that this greedy idea doesn't work. Find a counterexample for this algorithm, where your friend's seemingly reasonable greedy idea does not work, by describing an integer array. Also describe what intervals the provided pseudocode returns. Additionally, describe a better set of intervals (i.e. a way to cover all the positive elements of the array using less disjoint positive intervals than the greedy solution). You do not need to explain how you found your solution.

(b) If the idea doesn't work, then your friend's proof must be wrong! Clearly describe the error. A description cannot just be "the counter-example shows it must be wrong." You need to find an incorrect leap in logic, invalid assumption, missing case, or similar error. Nonetheless, it will probably help to run the algorithm on your example with an eye toward the proof to see where it breaks down.
Our explanation is only a few sentences, but it's ok if yours is a bit longer.

4. Beans [25 points]

As the lucky millionth customer of your local bean store, the owners have allowed you to fill up a bucket with any beans for free. They give you a comically-large bucket, so you don't need to worry about volume, but weight may be a problem: you decide you'll only be able to carry up to b pounds of beans. At the bean store, there are n different types of beans to choose from and bean i costs c_i dollars per gallon and weighs w_i pounds per gallon. Unfortunately, because of supply chain issues, the store is running low on beans. Each type has s_i gallons currently in stock. Note that since you're dealing with beans, you can fill your bucket with non-discrete quantities of beans (e.g., 0.5 gallons or $\sqrt{2}$ gallons). You wish to fill this bucket with the highest cost combination of beans that won't go over your weight limit.
As former algos students themselves, the store owners clue you in that a greedy algorithm might work. In this problem you will describe some "greedy rules" to summarize algorithm ideas. A "greedy rule" is a clear description someone could follow to select your beans (e.g., an ordering of bean types and rule for how much to include of each).[1]

(a) Describe a greedy rule that would **not** produce the optimal value. Present a counterexample where the greedy rule chooses a suboptimal assortment of beans, also give both the selection made by the greedy algorithm and the better selection.

(b) Describe a greedy rule that would produce the optimal value. Present a proof showing that the greedy algorithm with the described rule always chooses an optimal assortment of beans.

[1] For an example: "Sort the edges in increasing order and add an edge as long as it doesn't create a cycle" is a greedy rule for MSTs.

5. Extra Credit: Smarter Greedy

You are not required to submit attempts at extra credit problems (they do not count toward the dropped/counted problems at the end of the quarter). They will have significantly smaller effects on grades than the main problems, so we do not recommend attempting it until you've done all the other problems on the assignment. At the end of the quarter, after determining grade breaks, we will add in extra credit points. Note that we grade extra credit quite harshly and don't give much partial credit; we'll expect proofs and disproofs to be clear. Just getting the 'yes'/'no' of whether it works won't get partial credit without a good associated proof.

Here's another greedy algorithm for positive interval covering. Defined in Problem 3

```
function PositiveCovering(A[1..n])
    startInd ← 1
    while There is a positive element in A at startInd or later do
        while A[startInd] ≤ 0 do                    ▷ Find first uncovered positive element
            startInd++
                                                    ▷ Find rightmost valid end of the interval (store in endInd)
        endInd ← −1
        sum ← 0
        for curr from startInd to n do
            sum+ = A[curr]
            if sum > 0 then
                endInd ← curr
        Add startInd...endInd to list of Intervals
        startInd ← endInd + 1
    return All Intervals Found
```

Does this one work? If so, provide a proof. If not provide a counter-example and an explanation of why it is a counter-example.