Math 590 Homework 7 Fall 2019

Due: **Thursday**, November 21

1. **Ski Optimization!**[1]

   You have been hired as a consultant at the famed Forty-Two Black Diamonds Ski Resort. They will let you ski all winter for free in exchange for helping their ski rental shop with an algorithm to assign skis to skiers.

   Ideally, each skier should obtain a pair of skis whose height matches his or her own height exactly. Unfortunately, this is generally not possible. We define the **disparity** between a skier and his or her skis to be the absolute value of the different between the height of the skier and the pair of skis. **Your objective is to find an assignment of skis to skiers that minimizes the sum of the disparities between all of the skiers and their skis**.

   The input to this problem is an array of $n$ skiers (represented just by their heights) and an array of $m \geq n$ pairs of skis (each ski pair is represented as just the height of the skis). These arrays are given in sorted order from shortest to tallest. For example, we might have (in inches):

   ```
   skiers = [61 67 73]
   skis = [58 62 66 68 72]
   ```

   (a) [1pt] The infamous Prof. I.M. Rong from the Pasadena Institute of Technology is trying to steal your job. He proposes the following simple and fast 'greedy' algorithm for the problem:

   'The skiers and skis are already sorted from shortest to tallest. So, consider the skiers one-by-one from shortest to tallest. For each skier under consideration, assign that skier the pair of skis that most closely match that skier's height. Then remove that skier and pair of skis from consideration and repeat the process.'

   Find a small example where Prof. I.M. Rong's algorithm gives a solution that is worse than the optimal solution. You will need to provide a particular small set of skier heights, a small set of ski heights, show the solution that Prof. I.M. Rong's algorithm would produce, and then show a solution that is better.

   ANSWER: Consider the set:

   ```
   skiers = [60 65]
   skis = [55 61]
   ```

   According to Prof. I.M. Rong's algorithm, we would first consider the skier of height 60 and assign the skis of height 61. This leave the skis of height 55 for the

[1] Adapted from problem sets created by Harvey Mudd College CS Professor Ran Libeskind-Hadas.

skier of height 65. This solution has a disparity of $1+10=11$. It is clearly better in this case to give the height 55 skis to the height 60 skier, and the height 61 skis to the height 65 skier, for a disparity of $5+4=9$.

(b) [1pt] Often, we need to infer (and prove) a mathematical fact about our problem that we can exploit in developing our algorithm. Another consultant, Prof. Sue Persmart, made the following conjecture: If we have a short person and a tall person, it is never better to give the shorter person a taller pair of skis than were given to the tall person (we will call this an *inversion*). Your task is to prove this conjecture!

Specifically: Let $x$ and $y$ be the heights of two skiers with $x < y$ and let $s_x$ and $s_y$ be the heights of the skis with $s_x > s_y$. We'd like to show that if the person with height $x$ is given the skies with height $s_x$ and the person with height $y$ is given the skies with height $s_y$, then switching the skis between these two skiers would not increase the disparity (i.e., removing the inversion does not make the solution any worse).

One way to do this is to break the analysis up into a number of cases such as the case $x < y < s_y < s_x$, or the case $x < s_y < s_x < y$, etc. If you choose to use this approach, list the cases that you would need to consider and then analyze **just one** of these cases. (This is intended to save you time, because we trust that if you can do one case, you could do them all!)

ANSWER: There are six cases to consider:

$$x < s_y < s_x < y$$
$$x < s_y < y < s_x$$
$$x < y < s_y < s_x$$
$$s_y < x < s_x < y$$
$$s_y < s_x < x < y$$
$$s_y < x < y < s_x$$

Let's consider the case where $x < y < s_y < s_x$. In this case, when $x$ is given $s_x$ and $y$ is given $s_y$, the disparity is

$$|s_x - x| + |s_y - y|.$$

If the skis were switched, then the disparity would be

$$|s_x - y| + |s_y - x|.$$

Now, since $x < y < s_y < s_x$, we know that all four quantities $s_x - x$, $s_y - y$, $s_x - y$, and $s_y - x$ are positive. Therefore, both of the above disparities simplify to

$$|s_x - x| + |s_y - y| = s_x + s_y - x - y = |s_x - y| + |s_y - x|.$$

Therefore swapping the skis did not increase the disparity.

We can do this for all six cases, and so we can conclude that removing an inversion never leads to a worse solution. This tells us that there is an optimal solution that does not have an inversion. If we have an optimal solution that has inversions, we can remove those inversions without making the solution worse, and would therefore still have an optimal solution (but without inversions).

(c) [1pt] Assume that there are $n$ skiers and $n$ pairs of skis. Describe (in English) a fast algorithm for assigning skis to skiers, briefly explain how the proof of correctness would work (using the result from part (b)), and give the running time of your algorithm.

For part (d), you may assume that you have a function that correctly implements this algorithm and optimally assigns the $n$ skis to the $n$ skiers, returning the optimal disparity:

```
partC(skiers[0:n], skis[0:n])
```

ANSWER: From part (b), we concluded that we can always find an optimal solution without inversions. Note that when there are the same number of skis and skiers, there is only one solution without inversions: assign the smallest skis to the smallest skier, the second smallest skis to the second smallest skier, etc. This will take $O(n)$ time.

Note: we could also have assigned the largest ski to the largest skier, etc. This would have given the same assignment since there are equal numbers of skis and skiers.

(d) Finally, consider the general case that $m \geq n$.

i. [3pts] In simple and clear pseudo-code or English, describe a recursive algorithm for solving this problem. For now, assume that 'solving' means just returning the number which is the sum of the disparities in an optimal solution. Make sure to describe the base cases and the recursive call(s).

ANSWER: In my recursive function, I will consider the smallest ski and the smallest skier. I will then decide whether to assign that ski to that skier (use-it), or skip that pair of skis (lose-it). Note that we cannot skip a skier.

```
def assignSkis(skiers, skis, i, j):
    # Returns the minimum disparity when
    # assigning skiers[i:n] and skis[j:m].

    # Note: n = len(skiers) and m = len(skis).
    n = len(skiers)
    m = len(skis)

    # No more skiers who need skis, so 0 more disparity.
    if i = n:
        return 0

    # Same number of skis and skiers remaining, return from part (c).
    elif n-i == m-j:
        partC(skiers[i:n], skis[j:m])

    # Go to use-it or lose-it strategy.
    else:
        # Use-it: assign the shortest skis to the shortest skier and
        # see what happens.
        # Note that we have to add the disparity from this match.
        useIt = |skiers[i]-skis[j]| + assignSkis(skiers, skis, i+1, j+1)

        # Lose-it: don't assign the shortest skis to the shortest skier.
        # Note that it is never better to assign these skis to a taller
        # skier since that would result in an inversion.
        # So this case is: don't use the shortest skis at all.
        loseIt = assignSkis(skiers, skis, i, j+1)

        # Return whichever case gave the smallest disparity.
        return min(useIt, loseIt)
```

ii. [2pts] Next, describe how you would implement this algorithm using dynamic programming. In particular, describe what the DP table looks like and the order in which the cells would be filled in. A picture may help (although the picture alone will not be graded). For those of you interested in practical implementation, it may be informative to write the actual DP pseudo-code for filling the table.

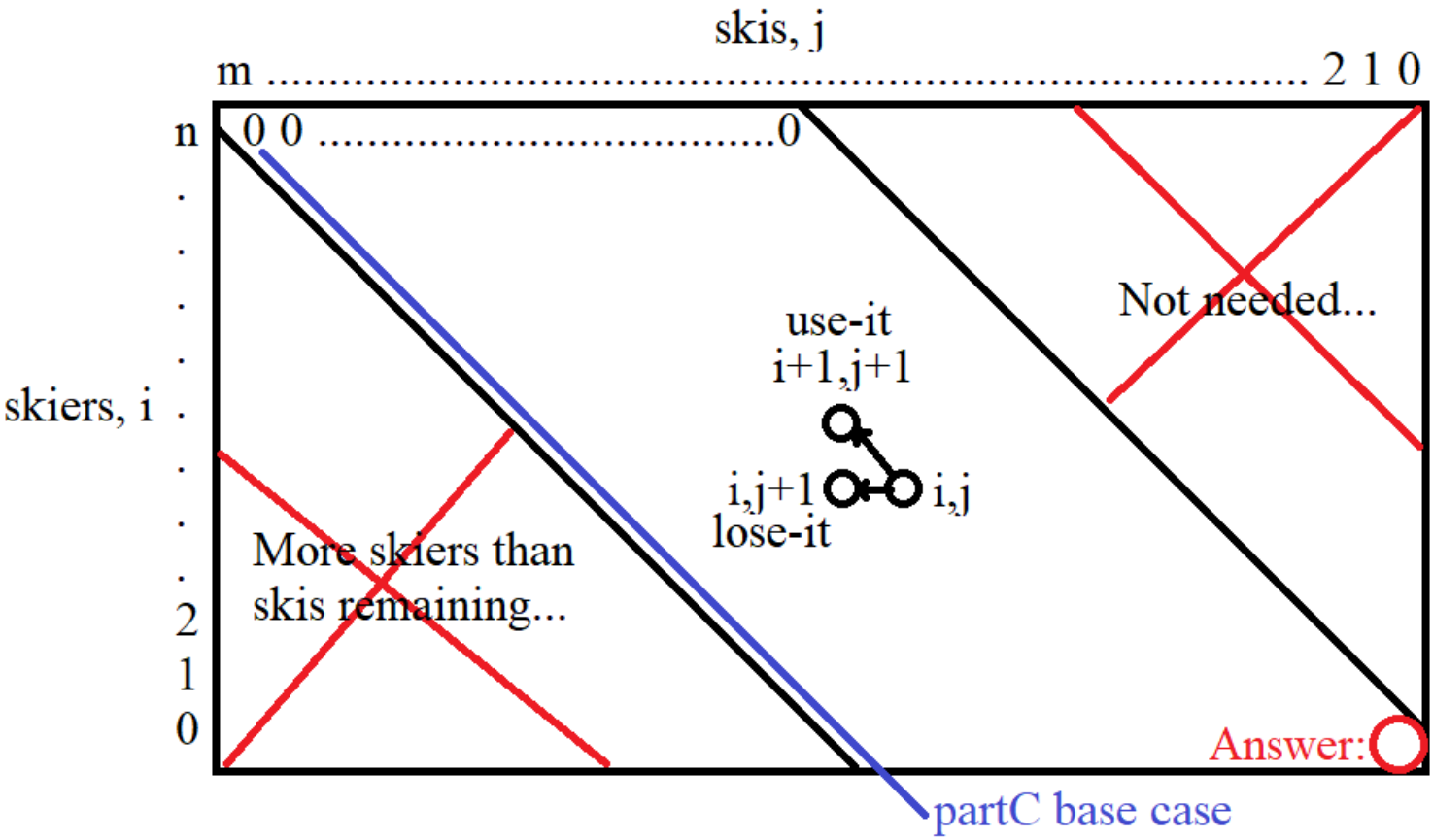
Note: there are dozens of ways to setup this table. I chose this method because I like having the solution live in the bottom right corner.

ANSWER: Put skis across the top and skiers down the side **(in reverse order, since we are removing from the front rather than from the back - so $i+1$ is up and $j+1$ is left)**.

Also, remember that the remaining skiers are `[i:n]` and the remaining skis are `[j:m]`.

- Everything below the main diagonal from the upper-left corner does not have to be considered, because those correspond to cases where there are more skiers than skis remaining, which is not allowed.
- The top row is a base case where there are no skiers, so fill it with 0.
- The main diagonal from the upper-left corner is a base case where there are the same number of skiers and skis remaining. So this diagonal can be filled using `partC(skiers[0:n], skis[0:n])`, which doesn't require any recursive calls. It will, however, require $k$ work for the $k$th element in this diagonal (assigning $k$ skis to $k$ skiers and summing the $k$ different disparities).
- For any given square in our table, we will either need the value directly left (the lose-it case with one fewer set of skis, since $j+1$ is left) or the value diagonal to the upper-left (the use-it case with one fewer set of skis and one fewer skiers, since $i+1$ is up and $j+1$ is left).
- We have the top row and the main diagonal filled from base cases, so we can now fill in the table one row at a time, left-to-right, starting with second row.
- We could also fill in the table one column at a time, left-to-right, starting with second column.
- It is slightly better to fill in the table one diagonal at a time, left-to-right. This way, we can stop when we reach the bottom right element and avoid an extra $n^2/2$ elements.

My table will look like this:




(continued on next page)

iii. [1pt] What is the running time of your algorithm?

ANSWER: We have two parts of the algorithm to count: filling in the base cases, and filling in the rest of the table.

Note that the base cases along the first row simply take $n$ work to fill in the $n$ zeros, but the base cases along the diagonal require $k$ work to compute the $k$th entry on this diagonal. So the total work for filling in the base cases is:

$$n + (1 + 2 + \cdots + n) \in O(n^2).$$

Note that this can actually be done faster by using the use-it (no lose-it option for this diagonal) approach to iteratively fill in the main diagonal, taking $O(n)$ time instead.

Now consider the work to fill in the rest of the table. If we filled across rows or down columns, we would have to fill in $mn - n^2/2 + n =\in O(mn)$ cells, each taking constant time, for a total of $O(mn)$ time.

If we filled down a diagonal, we would have skipped an extra $n^2/2$ cells. So now we would have to fill $mn - n^2$ cells. Along with the time to fill the main diagonal (quickly), this gives a total work of $mn - n^2 + n$. We can now make one an observation: we have been guaranteed that there are always more skis than skiers, and so $m \geq n$. Therefore $mn \geq n^2$, and so the overall runtime is $O(mn)$. This is not always a tight bound, because if $m = n + c$, the runtime would give

$$mn - n^2 + n = n^2 + cn - n^2 + n = (c + 1)n \in O(n).$$

Very technically, the tight bound we should report is:

$$O(mn - n^2).$$

This trick will not work if we filled by rows or columns, because the runtime was $mn - n^2/2 + n$, and so the $n^2$ terms wont actually cancel.

iv. [1pt] Briefly, describe how you could reconstruct an actual optimal solution matching skiers with skis using your DP table.

ANSWER:

- If the value in a given cell is the same as the value in the cell immediately to its left, then we were in the lose-it case. Else, the use-it case.
- If we were in the lose-it case, then the skis for that column were not used at all. Skip those skis and move on to the cell directly to the left.
- If we were in the use-it case, then the skis for that column were assigned to the skier from that row. Assign those skis to that skier, and move on to the cell diagonally to the upper-left.