Dynamic programming problems can be solved using implicit memoization. Here, you are expected to solve the dynamic programming using explicit memoization (i.e., build the appropriate tables, and fill them in the right order, etc). In the exams, you would probably be asked to solve dynamic programming using explicit memoization. For homework 5 you can still use implicit memoization, but this would not be acceptable later in the course.

Lenny Rutenbar, the founding dean of the new Maximilian Q. Levchin College of Computer Science, has commissioned a series of snow ramps on the south slope of the Orchard Downs sledding hill[1] and challenged Bill Kudeki, head of the Department of Electrical and Computer Engineering, to a sledding contest. Bill and Lenny will both sled down the hill, each trying to maximize their air time. The winner gets to expand their department/college into both Siebel Center and the new ECE Building; the loser has to move their entire department/college under the Boneyard bridge next to Everitt Lab.

Whenever Lenny or Bill reaches a ramp *while on the ground*, they can either use that ramp to jump through the air, possibly flying over one or more ramps, or sled past that ramp and stay on the ground. Obviously, if someone flies over a ramp, they cannot use that ramp to extend their jump.

1 Suppose you are given a pair of arrays $Ramp[1\,..\,n]$ and $Length[1\,..\,n]$, where $Ramp[i]$ is the distance from the top of the hill to the ith ramp, and $Length[i]$ is the distance that any sledder who takes the ith ramp will travel through the air.

Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air.

Solution:

To simplify boundary cases, let's add a sentinel ramp at the bottom of the hill with $Ramp[n+1] = \infty$.

For any index i, let $Next(i)$ denote the smallest index j such that $Ramp[j] > Ramp[i] + Length[i]$. Because the array $Ramp$ is sorted, we can compute $Next(i)$ for any index i in $O(\log n)$ time using binary search.

Now let $MaxAir(i)$ denote the maximum distance any sledder can spend in the air starting on the ground at the ith ramp. We need to compute $MaxAir(1)$ This function satisfies the following recurrence:

$$MaxAir(i) = \begin{cases} 0 & \text{if } i > n \\ \max\left\{\begin{array}{c} MaxAir(i+1) \\ Length[i] + MaxAir(Next(i)) \end{array}\right\} & \text{otherwise} \end{cases}$$

We can memoize this function into an a one-dimensional array $MaxAir[1\,..\,n+1]$, which we can fill from right to left.

```
MaxAir($Ramp[1..n], Length[1..n]$):
$Ramp[n+1] \leftarrow \infty$
$MaxAir[n+1] \leftarrow 0$
for $i \leftarrow n$ down to 1
    $next \leftarrow$ BinarySearch($Ramp,\ Ramp[i] + Length[i]$)
    $MaxAir[i] \leftarrow \max\{MaxAir[i+1],\ Length[i] + MaxAir[next]\}$
return $MaxAir[1]$
```

[1]The north slope is faster, but too short for an interesting contest.

Because of the binary search for *Next*(i) (here stored in the variable *next*), the algorithm runs in $\boldsymbol{O(n \log n)}$ ***time***.

Solution:

A completely different solution. Let us compute all the critical values for our problems. They are clearly $x_i = \mathit{Ramp}[i]$ and $y_i = \mathit{Ramp}[i] + \mathit{Length}[i]$, for $i = 1, \ldots, n$. We also add special values $s = 0$ and $t = +\infty$ to this set. Let us sort these values from left to right. Let Z be the resulting set of values. We now build a graph on top of these values. For $\alpha, \beta \in Z$, such that $\alpha < \beta$ and α, β are consecutive in the sorted order, we add the edge (α, β) to the graph. We put weight 0 on such edges. We also put edges (x_i, y_i), for all i, with weight *Length*[i]. This results in a directed graph G, that goes from left to right, and thus has no cycles. We are now looking for the longest weighted path from s to t in G. If $Z = \langle z_1 < z_2 < \ldots < z_m \rangle$ (here, $m = 2n + 2$), with $z_1 = s$ and $z_m = t$, then let $f(z_i)$ be the longest weighted path starting at z_i and ending at t. We have the following

$$\begin{aligned} f(z_m) &= 0 \\ f(z_i) &= \max_{(z_i, z_j) \in \mathsf{E}(\mathsf{G})} \Big(w\,(z_i, z_j) + f(z_j) \Big), \end{aligned}$$

where $w\,(z_i, z_j)$ is the weight assigned to the edge (z_i, z_j). We are interested in computing $f(z_1)$, which can be computed in $O(m)$ time using memoization. Here, we are using the property that every edge of the graph is inspected exactly once, and thus we can charge the relevant work to it.

As such, this problem is equivalent to the more general problem of computing the longest path in a DAG starting from a specific vertex (i.e., s), to another specific vertex (i.e., t).

The running time is dominated by sorting, which takes $O(n \log n)$ time.

2 Uh-oh. The university lawyers heard about Lenny and Bill's little bet and immediately objected. To protect the university from either lawsuits or sky-rocketing insurance rates, they impose an upper bound on the number of jumps that either sledder can take.

Describe and analyze an algorithm to determine the maximum total distance that Lenny or Bill can spend in the air *with at most k jumps*, given the original arrays *Ramp*[1 .. n] and *Length*[1 .. n] and the integer k as input.

Solution:

Again, add a sentinel ramp $\mathit{Ramp}[n+1] = \infty$, and for any index i, let *Next*(i) denote the smallest index j such that $\mathit{Ramp}[j] > \mathit{Ramp}[i] + \mathit{Length}[i]$.

Now let $\mathit{MaxAir}(i, \ell)$ denote the maximum distance any sledder can spend in the air, starting on the ground at the ith ramp, using at most ℓ jumps. We need to compute $\mathit{MaxAir}(1, k)$. This function obeys the following recurrence:

$$\mathit{MaxAir}(i, \ell) = \begin{cases} 0 & \text{if } i > n \text{ or } j = 0 \\ \max \left\{ \begin{array}{c} \mathit{MaxAir}(i+1, \ell) \\ \mathit{Length}[i] + \mathit{MaxAir}(\mathit{Next}(i), \ell - 1) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a two-dimensional array $\mathit{MaxAir}[1 \,..\, n+1, 0 \,..\, k]$, which we can fill by considering rows from bottom to top in the outer loop and filling each row in arbitrary order in the inner loop.

```
MaxAir(Ramp[1 .. n], Length[1 .. n], k):
  $Ramp[n+1] \gets \infty$
  for $\ell \gets 0$ to $k$ do
      $MaxAir[n+1, \ell] \gets 0$
  for $i \gets n$ down to 1
      $next \gets$ BinarySearch($Ramp$, $Ramp[i] + Length[i]$)
      for $\ell \gets 0$ to $k$
          $MaxAir[i, j] \gets \max\{MaxAir[i+1, \ell],\ Length[i] + MaxAir[next, \ell - 1]\}$
  return $MaxAir[1, k]$
```

Because we perform the binary search for $Next(i)$ outside the inner loop, the algorithm runs in $\boldsymbol{O(n \log n + nk)}$ ***time***.

3 **To think about later:** When the lawyers realized that imposing their restriction didn't immediately shut down the contest, they added a new restriction: No ramp can be used more than once! Disgusted by the legal interference, Lenny and Bill give up on their bet and decide to cooperate to put on a good show for the spectators.

Describe and analyze an algorithm to determine the maximum total distance that Lenny and Bill can spend in the air, each taking at most k jumps (so at most $2k$ jumps total), and with each ramp used at most once.

Solution:

```
MaxAir2(Ramp[1 .. n], Length[1 .. n], k):
  $Ramp[n+1] \gets \infty$
  $Length[n+1] \gets 0$
  for $i \gets n+1$ down to 1
      $next \gets$ BinarySearch($Ramp$, $Ramp[i] + Length[i]$)
      for $j \gets n+1$ down to $i$
          for $\ell \gets -1$ to $k$
              for $m \gets -1$ to $k$
                  if $\ell < 0$ or $m < 0$
                      $Air[i, j, \ell, m] \gets -\infty$
                  else if $i = n+1$ and $j = n+1$
                      $Air[i, j, \ell, m] \gets 0$
                  else if $i = j$
                      $Air[i, i, \ell, m] \gets \max\left\{\begin{array}{c} Air[i, i+1, m, \ell] \\ Length[i] + Air[i+1, next, m, \ell-1] \end{array}\right\}$
                  else
                      $Air[i, j, \ell, m] \gets \max\left\{\begin{array}{c} Air[i+1, j, \ell, m] \\ Length[i] + Air[next, j, \ell-1, m] \end{array}\right\}$
                  $Air[j, i, m, \ell] \gets Air[i, j, \ell, m]$
  return $Air[1, 1, k, k]$
```

Again, add a sentinel ramp $Ramp[n+1] = \infty$, and for any index i, let $Next(i)$ denote the smallest index j such that $Ramp[j] > Ramp[i] + Length[i]$.

Let $\text{MaxAir2}(i, j, \ell, m)$ denote the maximum time that Lenny and Bill can spend in the air if Lenny starts at ramp i, Bill starts at ramp j, Bill did not jump from ramps i through $j - 1$ (so Lenny still can use any of those ramps), Lenny has ℓ jumps remaining, and Bill has m jumps remaining. (Whew!) We develop a recurrence for this function as follows:

- The recurrence is based on Lenny's decision whether or not to jump at ramp i.
- If Bill and Lenny are at the same ramp i, and Lenny decides to jump, then Bill must sled down to ramp $i + 1$. Otherwise, Bill stays at ramp j.
- If Lenny ever sleds or jumps ahead of Bill (that is, if $i > j$), then (for purposes of computation) Lenny and Bill swap identities. In particular, if Lenny and Bill ever find themselves at the same ramp, then no matter what Lenny decides, Bill and Lenny will swap. Thus, "Bill" always means the sledder further downhill, and "Lenny" always means the sledder further uphill.

This function obeys the following recurrence:

$$\text{MaxAir2}(i, j, \ell, m) = \begin{cases} \text{MaxAir2}(j, i, m, \ell) & \text{if } i > j \\ -\infty & \text{if } \ell < 0 \text{ or } m < 0 \\ 0 & \text{if } i > n \\ \max\left\{ \begin{array}{c} \text{MaxAir2}(i, i+1, m, \ell) \\ \textit{Length}[i] + \text{MaxAir2}(i+1, \textit{Next}(i), m, \ell - 1) \end{array} \right\} & \text{if } i = j \leq n \\ \max\left\{ \begin{array}{c} \text{MaxAir2}(i+1, j, \ell, m) \\ \textit{Length}[i] + \text{MaxAir2}(\textit{Next}(i), j, \ell - 1, m) \end{array} \right\} & \text{otherwise} \end{cases}$$

We can memoize this function into a four(!)-dimensional array $\textit{Air}[1 \,..\, n+1, 1 \,..\, n+1, -1 \,..\, k, -1 \,..\, k]$. Each entry $\textit{Air}[i, j, \ell, m]$ with $i \leq j$ depends only on entries $\textit{Air}[i', j', \ell', m']$ where either $i' > i$, or $i' = i$ and $j' > i$. Thus, we can fill the array by decreasing i in the outermost loop, decreasing j in the next loop, and considering ℓ and m in arbitrary order in the inner two loops. The resulting algorithm (on the next page) runs in $\boldsymbol{O(n^2k^2)}$ ***time***.

(This is by far the most complicated dynamic programming algorithm we will see all semester!)