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

Describe and analyze an algorithm to determine the maximum total distance that Lenny and Bill can travel through the air. [Hint: Do whatever you feel like you wanna do. Gosh!]

**Solution:** The basic approach is almost identical to the dance-contest problem in Homework 5. To simplify boundary cases, we add a sentinel value $\text{Ramp}[n+1] = \infty$. (Intuitively, we add a “ramp” at the bottom of the hill, well beyond the end of any possible jump, and then end the race when Lenny and Bill reach this ramp.)

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

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

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

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

```plaintext
MaxAir(Ramp[1..n], Length[1..n]):
    Ramp[n+1] ← ∞ (sentinel)
    MaxAir[n+1] ← 0 (base case)
    for i ← n down to 1
        next ← BINARYSEARCH(Ramp, Ramp[i] + Length[i])
        MaxAir[i] ← max{MaxAir[i+1], Length[i] + MaxAir[next]}
    return MaxAir[1]
```

Because of the binary search for $\text{Next}(i)$ (here stored in the variable next), the algorithm runs in $O(n \log n)$ 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:** As in the previous problem, add a sentinel ramp $Ramp[n+1] = \infty$, and for any index $i$, let $Next(i)$ denote the smallest index $j$ such that $Ramp[j] > Ramp[i] + Length[i]$.

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

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

We can memoize this function into a two-dimensional array $MaxAir[1..n+1, 0..k]$, 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.

```plaintext
MaxAir(Ramp[1..n], Length[1..n], k):
    Ramp[n + 1] ← ∞
    for ℓ ← 0 to k
        MaxAir[n + 1, ℓ] ← 0
    for i ← n down to 1
        next ← BINARYSEARCH(Ramp, Ramp[i] + Length[i])
        for ℓ ← 0 to k
            MaxAir[i, ℓ] ← max{MaxAir[i + 1, ℓ], Length[i] + MaxAir[next, ℓ - 1]}
    return MaxAir[1, k]
```

Because we perform the binary search for $Next(i)$ outside the inner loop, the algorithm runs in $O(n \log n + nk)$ 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:** Again, add a sentinel ramp \( \text{Ramp}[n+1] = \infty \), and for any index \( i \), let \( \text{Next}(i) \) denote the smallest index \( j \) such that \( \text{Ramp}[j] > \text{Ramp}[i] + \text{Length}[i] \).

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 \( \ell \) jumps remaining, and Bill has \( m \) jumps remaining. (Whew!) We develop a recurrence for this function as follows:

- Without loss of generality, we assume \( i \leq j \). Intuitively, if Lenny ever sleds or jumps ahead of Bill (that is, if \( i > j \)), then (for purposes of computation) Lenny and Bill swap identities. Thus, “Bill” always means the sledder further downhill, and “Lenny” always means the sledder further uphill.
- 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 \( i + 1 \). (I missed this detail in the video!) Otherwise, Bill stays at ramp \( j \).

This function obeys the following recurrence:

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

We can memoize this function into a four-dimensional array \( \text{Air}[1..n+1, 1..n+1, -1..k, -1..k] \). Each entry \( \text{Air}[i, j, \ell, m] \) with \( i \leq j \) depends only on entries \( \text{Air}[i', j', \ell', m'] \) 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 \( \ell \) and \( m \) in arbitrary order in the inner two loops. The resulting algorithm (on the next page) runs in \( O(n^2k^2) \) time.
MaxAir2(Ramp[1..n], Length[1..n], k):
Ramp[n + 1] ← ∞
Length[n + 1] ← 0
for i ← n + 1 down to 1
    next ← BINARYSEARCH(Ramp, Ramp[i] + Length[i])
    for j ← n + 1 down to i
        for ℓ ← −1 to k
            for m ← −1 to k
                if ℓ < 0 or m < 0
                    Air[i, j, ℓ, m] ← −∞
                else if i = n + 1 and j = n + 1
                    Air[i, j, ℓ, m] ← 0
                else if i = j
                    Air[i, i, ℓ, m] ← max\{Air[i + 1, i, ℓ, m] \}
                        \{Length[i] + Air[i + 1, next, ℓ − 1, m]\}
                else
                    Air[i, j, ℓ, m] ← max\{Air[i + 1, j, ℓ, m] \}
                        \{Length[i] + Air[next, j, ℓ − 1, m]\}
                Air[j, i, m, ℓ] ← Air[i, j, ℓ, m]
        return Air[1, 1, k, k]