Some of your colleagues have decided to enter a sledding competition. In this competition, the contestants start at the top of a hill and race down on their sleds. This hill is filled with a number of ramps. Whenever a contestant 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.

The rules state that whoever spends the most time in the air wins the competition. After gathering details about the hills and ramps and doing some practice, your colleagues have turned to you to help them figure out how to get the first prize trophies in a few different categories.

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.

   For example, consider an instance where $n = 8$, $Ramp = [1,5,10,15,20,25,40,50]$ and $Length = [16,6,1,30,2,3,2,0]$. Then taking the first ramp will result in landing at distance 17 from the top of the hill, which is after the fourth ramp, and then taking all the subsequent ramps will give a total air time of $16+2+3+2 = 23$. On the other hand, if one skips the first one and takes the second ramp and the fourth one, the total air time is $6+30 = 36$.

   Describe and analyze an algorithm to determine the maximum total distance that a contestant can spend in the air.

Solution: To simplify boundary cases, we add a sentinel value $Ramp[n+1] = \infty$. (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 Nancy and Erhan reach this ramp.)

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 that one 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($Ramp[1..n]$, $Length[1..n]$):
  $Ramp[n+1] \leftarrow \infty$          ⟨⟨sentinel⟩⟩
  $MaxAir[n+1] \leftarrow 0$          ⟨⟨base case⟩⟩
  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 $MaxAir[1]$
```

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

2. Your colleagues have decided to try the expert version of the competition, which states that each contestant can use at most k jumps.

   Describe and analyze an algorithm to determine the maximum total distance that a contestant 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 ith ramp, using at most ℓ jumps. We need to compute $MaxAir(1, k)$. This function obeys the following recurrence:

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

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.

```
MaxAir($Ramp[1..n]$, $Length[1..n]$, $k$):
  $Ramp[n+1] \leftarrow \infty$
  for $\ell \leftarrow 0$ to $k$
      $MaxAir[n+1, \ell] \leftarrow 0$
  for $i \leftarrow n$ down to 1
      $next \leftarrow$ BinarySearch($Ramp$, $Ramp[i] + Length[i]$)
      $MaxAir[i, 0] \leftarrow 0$
      for $\ell \leftarrow 1$ to $k$
          $MaxAir[i,\ell] \leftarrow \max\{MaxAir[i+1,\ell],\ Length[i] + MaxAir[next, \ell-1]\}$
  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:** In the *team-based* expert version of the competition, sledders compete in teams of two. Each team member is still limited to k jumps, but each team can only use each ramp *once*, i.e., if a team member uses a ramp, then their teammate cannot use it.

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

Solution: Let us give our two contestants names, say, Erhan and Nancy.

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

To design a recurrence, let's consider what could happen at the ith ramp. There are four possibilities:

- If Erhan and Nancy are both on the ground at ramp i, we need to decide whether Erhan should jump at ramp i, or Nancy should jump at ramp i, or neither should jump at ramp i.
- If Erhan jumps over ramp i but Nancy does not, we need to decide whether Nancy should jump at ramp i.
- If Nancy jumps over ramp i but Erhan does not, we need to decide whether Erhan should jump at ramp i..
- If both Erhan and Nancy jump over ramp i, there is nothing to decide about ramp i.

Let $MaxAir2(i, j, \ell, m)$ denote the maximum time that Nancy and Erhan can spend in the air under the following conditions.

- If $i = j$, then both sledders are on the ground at ramp i. In this case, at most one of the sledders can jump at ramp i; any sledder that does not jump sleds down to ramp $i + 1$.
- If $i < j$, then Erhan is on the ground at ramp i, and Nancy is jumping over ramp i and will land just before ramp j. In this case, Erhan can either jump at ramp i or sled down to ramp $i + 1$.
- If $i < j$, then Nancy is on the ground at ramp j, and Erhan is jumping over ramp j and will land just before ramp i. In this case, Nancy can either jump at ramp j or sled down to ramp $j + 1$.
- Erhan has ℓ jumps remaining, and Nancy has m jumps remaining.

In the second and third cases, the airtime for the sledder in the air is *not* included in the total. We handle the case where both sledders are in the air over ramp i by moving down the hill to the first landing. (Whew!)

Formalizing our case analysis gives us the following recurrence:

$$
\textit{MaxAir2}(i,j,\ell,m) = \begin{cases}
-\infty & \text{if } \ell < 0 \text{ or } m < 0 \\
0 & \text{if } i > n \text{ and } j > n \\
\max\begin{Bmatrix} \textit{MaxAir2}(i+1,i+1,\ell,m) \\ \textit{Length}[i] + \textit{MaxAir2}(\textit{Next}(i),i+1,\ell-1,m) \\ \textit{Length}[i] + \textit{MaxAir2}(i+1,\textit{Next}(i),\ell,m-1) \end{Bmatrix} & \text{if } i = j \le n \\
\max\begin{Bmatrix} \textit{MaxAir2}(i+1,j,\ell,m) \\ \textit{Length}[i] + \textit{MaxAir2}(\textit{Next}(i),j,\ell-1,m) \end{Bmatrix} & \text{if } i < j \\
\max\begin{Bmatrix} \textit{MaxAir2}(i,j+1,\ell,m) \\ \textit{Length}[j] + \textit{MaxAir2}(i,\textit{Next}(j),\ell-1,m) \end{Bmatrix} & \text{if } i > j
\end{cases}
$$

We can memoize this function into a four-dimensional array $\textit{Air}[1..n+1, 1..n+1, -1..k, -1..k]$. Each entry $\textit{Air}[i,j,\ell,m]$ 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. To speed up evaluation, we precompute all values of $\textit{Next}(i)$ at the start. The resulting algorithm runs in ***$O(n^2k^2)$ time***.

```
MaxAir2(Ramp[1..n], Length[1..n], k):
  $Ramp[n+1] \leftarrow \infty$
  $Length[n+1] \leftarrow 0$
  for $i \leftarrow 1$ to $n$
      $Next[i] \leftarrow$ BinarySearch($Ramp$, $Ramp[i] + Length[i]$)
  for $i \leftarrow n+1$ down to 1
      for $j \leftarrow n+1$ down to $i$
          for $\ell \leftarrow -1$ to $k$
              for $m \leftarrow -1$ to $k$
                  if $\ell < 0$ or $m < 0$
                      $Air[i,j,\ell,m] \leftarrow -\infty$
                  else if $i = n+1$ and $j = n+1$
                      $Air[i,j,\ell,m] \leftarrow 0$
                  else if $i = j$
                      $Air[i,i,\ell,m] \leftarrow \max\begin{Bmatrix} Air[i+1,i+1,\ell,m] \\ Length[i] + Air[Next[i],i+1,\ell-1,m] \\ Length[i] + Air[i+1,Next[i],\ell,m-1] \end{Bmatrix}$
                  else if $i < j$
                      $Air[i,j,\ell,m] \leftarrow \max\begin{Bmatrix} Air[i+1,j,\ell,m] \\ Length[i] + Air[Next[i],j,\ell-1,m] \end{Bmatrix}$
                  else ⟨⟨$i > j$⟩⟩
                      $Air[i,j,\ell,m] \leftarrow \max\begin{Bmatrix} Air[i,j+1,\ell,m] \\ Length[i] + Air[i,Next[j],\ell,m-1] \end{Bmatrix}$
  return $Air[1,1,k,k]$
```

■