Solution for the Holiday Task

For ease of description, we will first describe a simple $O(n \log n)$-time solution for the special case of the starting city *start* being the one with the index 0 for any fixed d. That is, $start = 0$ and d is a given number. Then we extend this solution to build a table of solutions in $O(n \log^2 n)$ time for all possible values of d. Finally we describe how to extend this solution to solve the general case of an arbitrary *start* with the same asymptotic time complexity.

Simple solution for $start = 0$ and a given fixed d

Without lost of generality, assume we start at the leftmost city and move right. It is easy to see that we only need to move right and there is no need to move left at any time. Assume in an optimal solution, city *right* is the rightmost city we will travel to. Then we can visit up to $d - right$ cities among the cities with labels $0, 1, \ldots, right$. In order for the solution to be optimal, we want to visit the $d - right$ cities with the largest number of attractions. That is, if we sort the cities with labels $0, 1, \ldots, right$ using the number of attractions as their keys, then we want to know the sum of the $d - right$ largest number of attractions.

Segment tree We use a data structure called segment tree for this part though it may appear this data structure is not needed to solve this very special case. However, it will be clear why this data structure is used in the solution to the intermediate case. The segment tree data structure has been used in previous IOI contests including 2001 Baltic OI. The segment tree has many variations. We will use the following one. A segment tree is a rooted complete binary tree with leaves carrying a flag indicating whether this leaf is active or not, and a value. For each internal node v, it keeps the sum of the values of all of the active leaves in the subtree rooted at v. Each internal node also maintains the number of currently active leaves in the subtree rooted at v.

Assume the segment tree has n leaves. Note that we need to add dummy leaves of n is not a power of 2. Further assume the values of the leaves, active or in-active, are in non-increasing order from left to right. To maintain this data structure, it takes $O(\log n)$ time to turn on or off any leaf. It also takes $O(\log n)$ time to find out the sum of the values in the largest x active leaves for any given x. A side note is when x is more than the number of active leaves, then we simply output the sum of the values of all active leaves.

Algorithm Initially, we sort the cities using their number of attractions as keys in non-increasing order. Then in this order, we place them as leaves in the segment tree from left to right with all leaves in-active. The number of attractions are now the values of the leaves. The initialization phase takes $O(n \log n)$ time. We turn on a leaf when it is being move to during the search of our solution. We iterate on all possible values of the rightmost city we can move to. Hence it takes a total of $O(n \log n)$ time to find a solution for this easy special case.

Intermediate solution for $start = 0$ and all possible values of d

Now we describe how to solve this intermediate case. In our previous solution, we can find the maximum number of attractions we can visit given any d. Let $f(d)$ be the label of the city we move to in d days so that the maximum number of attractions can be found. Note that $f(d)$ may not be unique. In the case of multiple ones, we pick the one with the smallest label. We now want to build a table for all possible values of d. The idea is to use recursive divide-and-conquer approach. Let M be the maximum number of d. For ease of description, let M be a power of 2. To compute the solutions for $f(1), f(2), \ldots, f(M)$ we first find $f(M/2)$ using our previous algorithm by iterating through all cities from 0 to n and then recursively compute $f(1), f(2), \ldots, f(M/2-1)$ in one branch by considering only cities from 0 to $f(M/2)$, and $f(M/2+1), f(M/2+2), \ldots, f(M)$ in the other branch by considering only cities from $f(M/2)$ to n. In the branch of computing $f(1), f(2), \ldots, f(M/2-1)$, we first compute $f(M/4)$ among cities 0 to $f(M/2)$. In general, the total amount of time spent in each level of recursive calls takes a total of $O(n \log n)$. There are a total of $O(\log n)$ levels. Hence the overall time complexity is $O(n \log^2 n)$.

In solving this intermediate case, it is now clear how the segment tree is useful. First we only need to do the initialization once. Secondly, we can easily turn on or off a leaf to accommodate the fact that the cities that we pay attention to in each level of recursion. For example, only half of the leaves are active during the second level of recursion.

General solution for an arbitrary value of *start*

Now we are ready to show the general solution. Observe the fact that when the value of *start* is arbitrary, then the solution can be found in either one of the following 2 cases. We first move right to a city, then move left from this point, and then finally stop at a city left of *start*. Or we first move left to a

city, then move right from this point, and then finally stop at a city right of *start*. That is, we only change the direction once. Without lost of generality, we assume the first scenario. We first use the immediately solution to find $f(t)$ for all possible of values t. Then we also use the immediately solution to find $g(t)$ for all possible values of t where $g(t)$ is the city we will stop in an optimal solution if we only move left and the starting city is $start - 1$. We first iterate on d_0 the days we want to spend on moving and visiting cities to the right of, and including, *start*. Using the solution to the intermediate case, we know $f(d_0)$. Then we know we can spend $d - d_0 - (f(d_0) - start) - 1$ days on the cities to the left of *start*. Hence the overall time complexity is $O(n \log^2 n)$.