Jordan Allard
CSCI 264-01
Homework 4

HW4 Problem 2 Writeup

a) Heart of the Solution

WHAT: S[j] = the max weight of a set of non-overlapping intervals, including travel times, chosen from the first j intervals sorted by finishing times and ending with the jth interval.

HOW: S[j] = max S[k] + 1, k = 0 to j - 1

WHERE: max S[j]

b) Pseudocode

```
A = array of intervals (start, finish)
B = an 2D array of travel times between intervals
Sort A by A[i].finish from lowest to highest using MergeSort
maxTotalIntervals = 0
S[1] = 1
For j = 2 to n:
      currentMax = 0
      For k = 1 to j:
            If A[k].finish + B[k][j] <= A[j].start:
                  If S[k] > currentMax:
                        currentMax = S[k]
      S[j] = currentMax + 1
      If S[j] > maxTotalIntervals:
            maxTotalIntervals = S[j]
Return maxTotalIntervals
```

c) Proof of Correctness (Explanation of HOW)

If each total S[j] in the solution array is the maximum intervals taken that includes the jth interval, then the next solution can ignore the previous intervals and only needs to know the travel times of the current interval. If the current interval doesn't overlap with the next interval, the entire sequence can be used. Thus for each element of S[j], the algorithm only needs to find an interval that doesn't overlap with the current one and that has the largest number of previous valid intervals, and add the current interval to the count. The algorithm checks every valid non-overlapping interval so is guaranteed to find a valid maximum, and the travel time for each interval for determining validity is guaranteed to be accurate because each of the previously stored solutions at j ends with the jth interval.

d) Running Time Estimate

O(n^2)

e) Running Time Estimate Reasoning

The total time complexity of this algorithm can be broken down as follows:

- Populating the array of intervals from input --> O(n)
- Populating the 2D array of travel times from input --> O(n^2)
- Sorting the intervals by finish times with MergeSort --> O(n log n)
- Calculating the max intervals for each starting interval --> O(n^2)

Thus the total time complexity is O(n) + O(n^2) + O(n log n) + O(n^2) = O(n^2).