Homework 5, Programming Assignment, Algorithms, Spring 2020, Professor Li
David Russell

Programming Assignment Documentation:
This is an optimization program that finds the “maximum value” for a set of activities(intervals), where the activities do not overlap in terms of “starting” and “finishing” time, each activity being assigned a value that it is worth.

The program is designed to be run with 2 command line arguments as outlined in the Programming Assignment documentation provided by Dr. Li. As the document outlines, the first argument is meant to the be filename of the input to the program and the second argument is meant to be the filename of the output of the program

Analysis:
I tested my program with the inputs provided by Dr. Li and also some additional cases that I created myself, until I was satisfied that the results were consistent and reproducible. While the general program is that of finding the maximum value, there were additional conditions added that significantly increased the complexity of the problem.

The first such requirement was this: Generate the set of intervals, in descending order, that generate a maximum value.

This meant that in addition to using Dynamic Programming principles to calculate the maximum value for each relevant “time” in the total interval, I had to keep track of what sub intervals had contributed to the maximum value, and find a method for updating that list when the contributing intervals changed. I chose to keep track of the intervals contributing to the maximum value calculated for each “time” and use those to generate the final output in the same dynamic programming fashion.

The second such requirement was this: Identify whether there exist multiple solutions with the same maximum value.

This stumped me for a while because the dynamic programming approach, while more efficient than a brute force approach, did not lend itself to exploring every possibility, merely ensuring that the possibility it ended up with at the end was a member of a set of maximum values.

However, I realized that there was a very simple solution: run the same algorithm “backwards”, and this would necessarily discover the maximum value by other means, if they existed.

Asymptotic Analysis:

The sorting that takes place uses TimSort which is an optimized sorting algorithm based on InsertionSort and MergeSort with a worstcase run time of O(nlogn), therefore the dominant factor of the algorithm is the nested for loop, which operates on each unique start and end point, which, will be of no greater magnitude than 2*n, (one unique start and end for each n), and

inside of that loop, on each “sub-interval” n. Thus, the algorithm has a worst case analysis of O(n^2) and thus polynomial.