Project 2 Report

Amanda Shohdy | [amandashohdy@csu.fullerton.edu](mailto:amandashohdy@csu.fullerton.edu)

Edwin Rodriguez | [edwin.rodriguez@csu.fullerton.edu](mailto:edwin.rodriguez@csu.fullerton.edu)

Jasmine Rodriguez | [jasminerrodriguez@csu.fullerton.edu](mailto:jasminerrodriguez@csu.fullerton.edu)

Problem 1

Problem 1: Ensuring Convenient Schedules was intended to determine intervals of time when all members of a group are available for a meeting of a minimum duration. Two or more arrays representing slots that are already booked and the login/logout times of group members are taken as input. The program requires an array that represents an individual’s existing schedule, the daily working periods of group members, and an expected duration of the meeting. The output is a list of intervals when all members are available to meet.

This algorithm condenses the schedules for two members based on their availability and their set work time so that they may agree on a meeting time. Their availability in the algorithm is known as person one or person’s two schedules which means that this time the members cannot meet. The set work time is known as the daily\_act which is the full range of time the members are logged on. The duration of the meeting is there so that the meeting cannot be less than that amount. With those factors in mind, a list was created with those time intervals where the members can meet.

The first loop has 14 operations performed n times in the first loop. 9 operations performed n^2 times in the nested loop and 10 fixed number of operations. With the final worst case time being O(n^2).

Problem 2

Problem 2: Arraylist and Subsets was intended to determine a contiguous part of an array of numbers that produces the maximum possible sum. The program requires an array or list of integers to output the section of the array that results in the maximum sum. This problem is done using the exhaustive search approach.

The algorithm defines the function largest\_sum\_subarray(arr). This function stores values max\_sum, start, end, and n. For each index in the range of the array, for each possible interval starting at that index, the sum of the current interval is compared to the sum stored in max\_sum. If the current sum is greater than the max sum, the current sum becomes the max sum and the start and end indices are updated. After all possible intervals have been checked against the max\_sum, the max\_sum variable is returned and output.

The step count analysis of the efficiency class is determined to be O(n2 + 4). This belongs to O(n2). The nested for loops have an efficiency class of O(n2) and there are four steps of constant time. Thus, the final worst case time is O(n2).

Sample Input Largest Sum Sub-Array Outputs:

1.

Enter a list of integers separated by spaces: 10 2 -5 1 9 0 -4 2 -2

User input list: [10, 2, -5, 1, 9, 0, -4, 2, -2]

Largest sum subarray: [10, 2, -5, 1, 9]

2.

Enter a list of integers separated by spaces: -7 1 8 2 -3 1

User input list: [-7, 1, 8, 2, -3, 1]

Largest sum subarray: [1, 8, 2]

3.

Enter a list of integers separated by spaces: 9 7 2 16 -22 11

User input list: [9, 7, 2, 16, -22, 11]

Largest sum subarray: [9, 7, 2, 16]

4.

Enter a list of integers separated by spaces: 6 1 9 -33 7 2 9 1 -3 8 -2 9 12 -4

User input list: [6, 1, 9, -33, 7, 2, 9, 1, -3, 8, -2, 9, 12, -4]

Largest sum subarray: [7, 2, 9, 1, -3, 8, -2, 9, 12]