CS 325 HW4

Hao (Jeff) Deng 932912420

Problem 1). Class Scheduling

a). Algorithm

- Start
- Sort the classes according to their finishing time in a decreasing order
- Start scheduling by compare the class start time in the beginning of the list see if it's starting time is greater or equal to the finishing time of the next class
    - If so, schedule the class
    - Else, choose a new lecture hall when the class cannot be scheduled
- Repeat the comparing and scheduling steps
- End

b). Time

The running time of this algorithm should be O(n log n) because we will first need to take in n classes into an array or a list and then we will sort it using the least time complexity algorithm, in this case we can use merge sort which is O(n log n). After all the class placements, each class will take O(n) time.

So the complexity would be T(n) = O(n log n)

Problem 2). Scheduling jobs with penalties:

a). Algorithm

- Start
- Sort the jobs according to their penalties in a decreasing order
- Looping through the sorted array
    - If the deadline of the current job[i] deadline == i or if i == n
        - Then add this job to the slot at i
    - Else
        - Go through the loop again
- End

b). Time

The time complexity seems to be O(n log n) because first scanning the list to an array is a linear time, then sort the list with a O(n log n), and then print the sorted array out in a linear time. Therefore, overall the time complexity is T(n)= O(n log n)

Problem 3). CLRS 16-1-2 Activity Selection Last-to-Start

a). How this approach is a greedy algorithm?

Selecting the last activity to start is just a greedy algorithm but starting from the end rather from the beginning. Let's say we are given a set of A = $\{a_1, a_2, a_3....\}$ of activities and $a_i = [s_i, f_i]$ for start and finish time. We are asked to find the optimal solution by selecting the last activity to start, so let's create

another set B = {b_1 , b_2 , b_3……} where b_i = [f_i , s_i], that is b_i is the reverse of a_i . The subset of $a_i \subseteq$ is mutually compatible only if the corresponding subset $b_i \subseteq$ B is also mutually compatible. Therefore, an optimal solution for A indicates the optimal solution for B and vice versa.

b). Proof yield optimal solution

Since we have the proposed approach of selecting the last activity to start that is compatible with all its previous selected activities. When run it on A, gives the same answer as the greedy algorithm as it runs on B. The solution that we can find in the proposed approach for A is corresponds to the solution with the greedy algorithm that finds for B, therefore the solution is optimal.

Problem 4). Activity Selection Last-to-Start Implementation

a). Algorithm

- Start
- Sort the activities according to their starting time in a decreasing order
- Start selecting by compare the first activity start and the second activity's finish time to see if it's greater or equal to the second's finish time
    - If so, that means we can select this activity and add to the slot since there's no conflict
    - Else, ignore and continue the loop
- Repeat the comparing and selecting steps
- End

Then we can print the array out starting from behind since it was sorted in decreasing order.

b). Pseudocode

```
Selecting_activities(2d vector array, array_size){
    Calling mergeSort(array,0,array_size) //Using Merge sort for sorting the array in a decreasing order according to starting time after getting the values from the text file
    Create a temporary 2d vectory array called temp
    Push the first value from the array to the temp
    For i=1, i less than array_size{
        If the temp[temp.size() – 1][starting time] >= array[i][finishing time]
            Push the current array[i] to the temp array
        Else
```

```
                Continue the loop //that will ignore the conflicted activities
        }
}
```

c). Time

The time complexity for this algorithm would be O(n log n) because first, the program will take in n activities from the file, and then the program will sort that array using merge sort algorithm, then afterward, the program will go through the array and select the optimized activities and print them out. Therefore, the overall time complexity for this program would be T(n) = O(n log n).