CS 325 – Analysis of Algorithm

Name: Shou-Cheng Wu

Email: [wusho@oregonstate.edu](mailto:wusho@oregonstate.edu)

ID: 933 920 148

**Problem 1: Scheduling jobs with penalties:**

**a) Design a greedy algorithm to find a schedule such that all jobs are completed and the sum of all penalties is minimized. Provide a verbal description and pseudocode.**

**Verbal Description:**

In order to design a greedy algorithm with minimal penalties, the first step we are supposed to do is to sort the jobs by penalties in a decreasing order. Second, based on these sorted jobs, we start to consider how to put it in our schedule line. Our criterion is to put the larger penalty’s job earlier in the schedule line if it is possible. But, the position of this job in the schedule line is what we are really concerned about. My method is to put it at the last minute. For example, if has the largest penalty with a deadline equals 5, we will process it earlier and put it in the schedule line from 4 to 5. If is the second largest penalty with a deadline equals 5 as well, because it is also a big penalty, I want to put it in the schedule line as well. But the position I put in is from 3 to 4 because 4 to 5 in the schedule is not occupied. Then, do it so on so forth. If we have a job that cannot be put in our schedule line, just leave it and put it back when we complete the rest of jobs in our original sorted jobs.

For example,

|  |  |  |
| --- | --- | --- |
| job | Deadline | Penalty |
| 4 | 2 | 10 |
| 2 | 2 | 8 |
| 3 | 2 | 5 |
| 1 | 3 | 3 |

Step 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time line | 0-1 | 1-2 | 2-3 | 3-4 |
| Job |  | 4 |  |  |

(Because job4 has the highest penalty)

Step 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time line | 0-1 | 1-2 | 2-3 | 3-4 |
| Job | 2 | 4 |  |  |

Because job2 has the second largest penalty and time line 1 to 2 is occupied, move forward.

Step 3:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time line | 0-1 | 1-2 | 2-3 | 3-4 |
| Job | 2 | 4 | 1 |  |

Because there is no place for job3, we leave it and start to consider job1.

Step 4:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time line | 0-1 | 1-2 | 2-3 | 3-4 |
| Job | 2 | 4 | 1 | 3 |

Because there is nothing in our sorted job array, we start to put the jobs which we have not assigned.

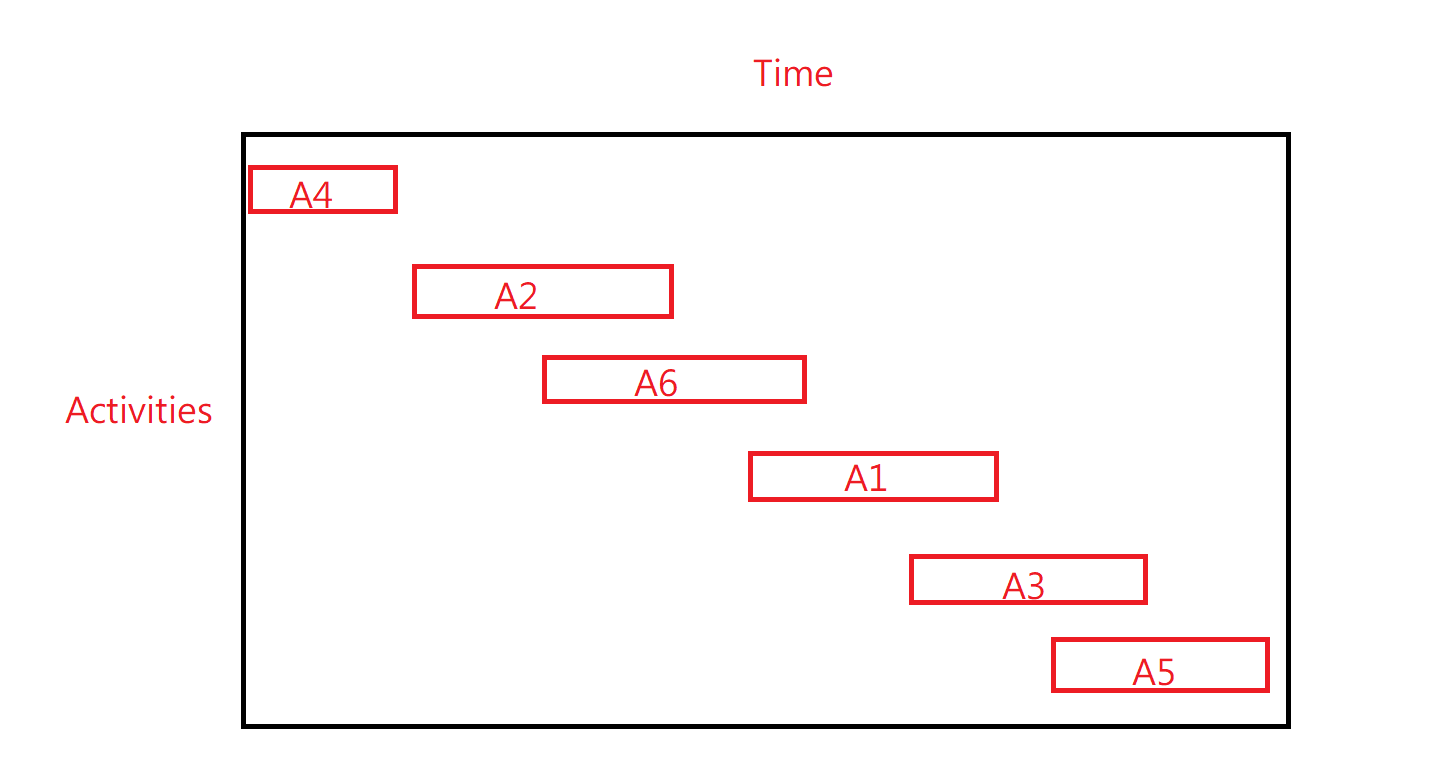
**b) What is the running time of your algorithm?**

The time complexity is O()

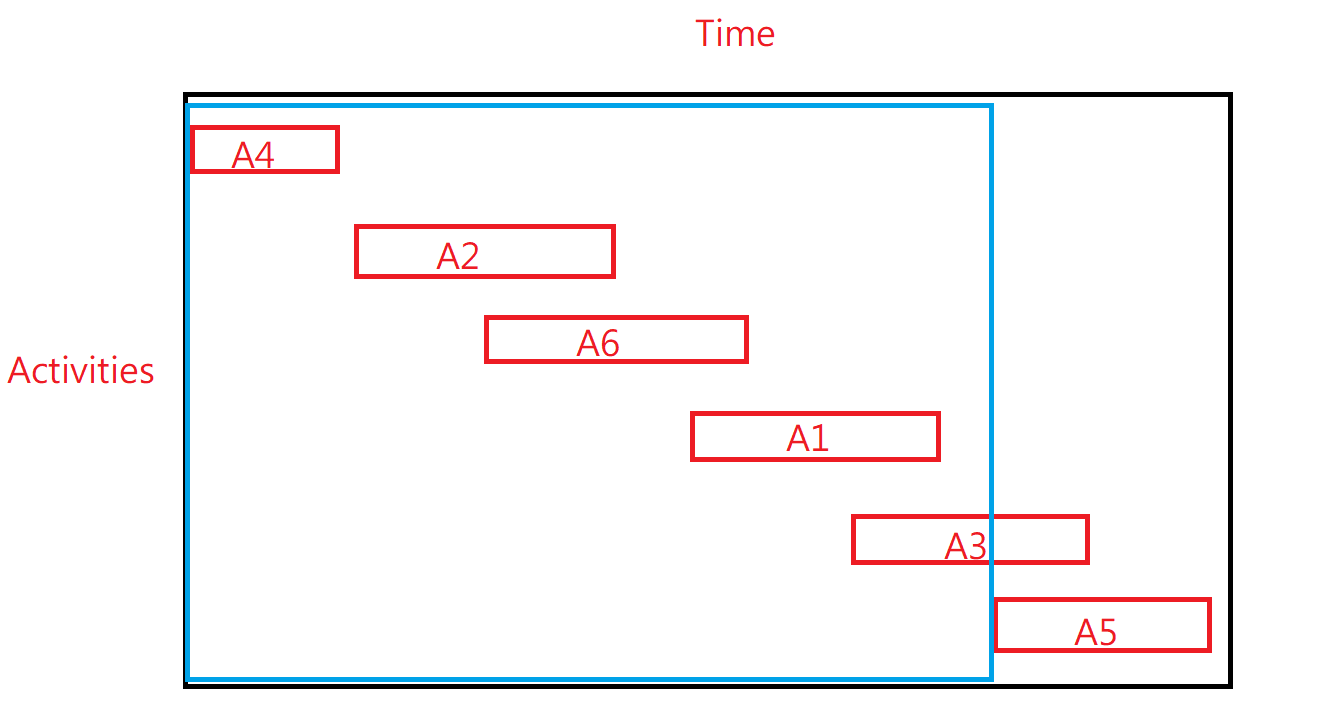
**Problem 2:** **CLRS 16-1-2 Activity Selection Last-to-Start**

Doing activity selection Last-to-Start by greedy algorithm, what we should do first is sorting the activities by start time in a decreasing time. If the next activity’s finish time is earlier than current activity’s start time, it means that this activity could be selected because there is no time conflict. It is a greedy algorithm because we are always doing the best selection at that moment. For example, the graph below is showing that we have sorted activities by finish time.

Because we want to select activities by the principle of last-to-start, we will pick A5.



Then, because we have already picked A5, the next acceptable activity is A1. It is because its finish time is earlier than A5’s start time. It means that it doesn’t have any conflicts. It is a greedy algorithm since we always pick the best option in the current case. More specifically, after we take A5, A1 is the best option in this case. Because we are always doing the best selection, it can be said as an optimal solution for the greedy algorithm.

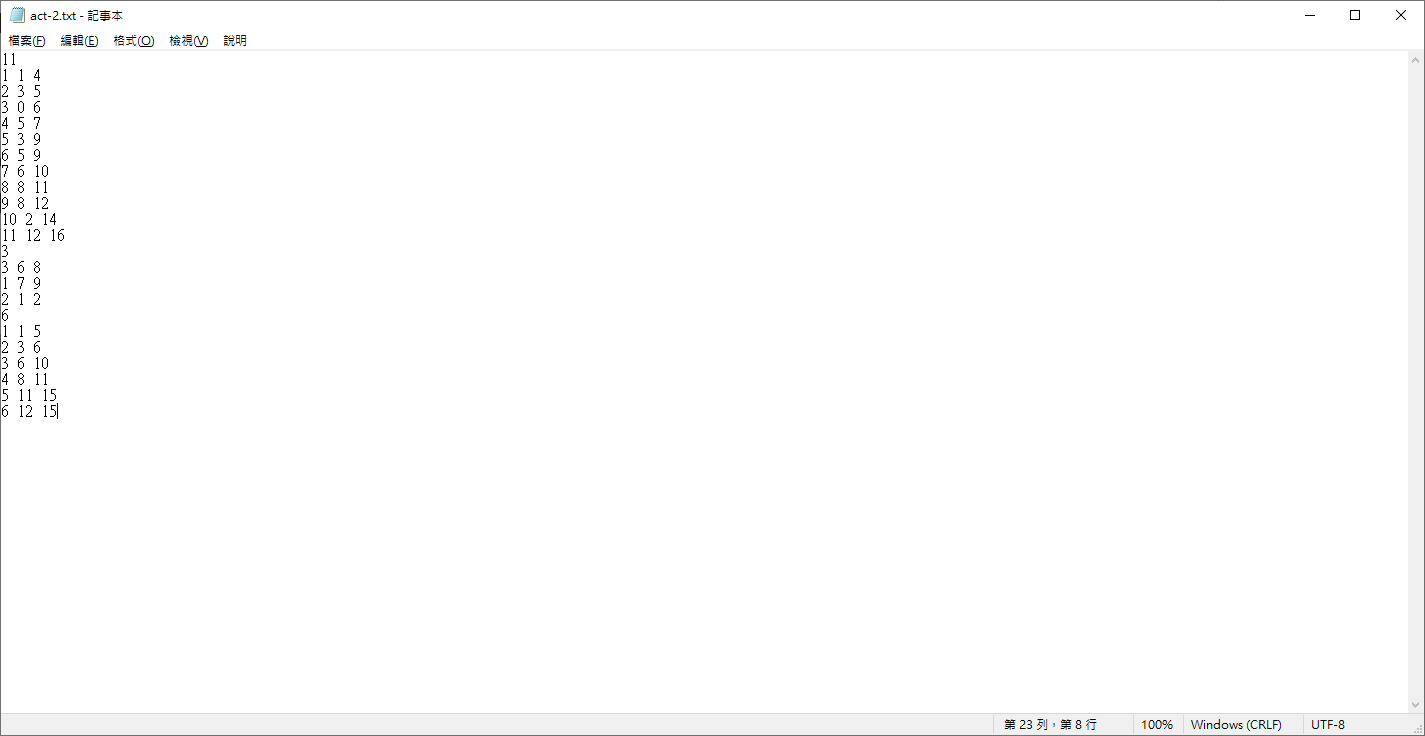


**Problem 3: Activity Selection Last-to-Start Implementation**

**Verbal description:**

The first step to solve this problem is to sort the data correctly. Therefore, except for the lines with only one element, I would store the activities’ information into a nx3 matrix. For example, in our act-2.txt, I would store 1 1 4, 2 3 5… as a matrix, [[1,1,4],[2,3,5]…] and store 11,3,6 in a vector, called set. Then, group the cases for future use. After I finish processing the data, I need to sort them by start time in a decreasing order. After we sort the data, we would put activities one by one. The criterion is the next finish time should be less than the current start time. It means it would not conflict with other activities in the schedule.

**The theoretical running time of this algorithm is** sorting time + greedy algorithm. The sorting algorithm I use here is insert sort, so the time complexity of it is O(). Therefore, the theoretical running time is O() + O(n) (greedy algorithm). If n is very large, we can say the running time of my activity selection last-to-start algorithm is O()



act-2.txt.