Problem 1. Class Scheduling

Our greedy algorithm will consider classes in increasing order of finish time. The algorithm will schedule classes in a hall only if the class is compatible with the classes already taken. If no more classes can be scheduled in the hall and there are still remaining classes left to be scheduled, the algorithm will schedule classes in a new hall by choosing the next class with the earliest finish time.

$Let\ C\ be\ the\ set\ of\ classes\ \{(s_i, f_i), 1 \le i \le n\}$ of start and finish times of n classes
$Let\ H\ be\ a\ set\ of\ schedules\ \{\{(s1, f1), (s2, f2) \ldots\}, \{(s3, f3), (s4, s5), \ldots\} \ldots\}$
//Each schedule in H will be composed of classes that are compatible
//The number of schedule will be equal to the number of halls required.
Type equation here.

$Class\_Scheduler\{$
  $Let\ c_i = (s_i, f_i), where\ s_i\ is\ the\ start\ and\ f_i\ is\ the\ finish\ of\ class\ i, 1 \le i \le n$
  $Let\ C = [c_1 \ldots . c_n]$
  $Sort(C)\ in\ order\ of\ increasing\ finish\ times$
  $Let\ H = [\,], an\ empty\ array$
  $while\ C\ is\ not\ empty$
    $Let\ T = [\,]\ be\ an\ empty\ array$
    $T[1] = C[1]$
    $for\ i \leftarrow 2\ to\ C.length$
      $if\ s_i \ge f_k:$
        $T.append(c_i)$
        $Delete\ C[i]$
    $H.append(T)$
$\}$

Sort will take O(n log n), if merge sort is used. In the worst case n halls will be needed, the outer while loop will execute n times and the inner for loop will execute n times. Deletion and shifting of the C[ ] is O(n).

$$T(n) = O(n\,logn + n^2 + n) \therefore T(n) = O(n^2)$$

2. Scheduling Jobs with penalties.

Our greedy algorithm will first sort jobs in order of decreasing penalties. We will define an array S[] = [1.....n], where each index represents a one minute time slot. The algorithm will then seek to schedule the job in latest possible spot without exceeding the deadline. It search the possible intervals $1 \leq k \leq d_i$. If no spot is available it will search from the end of the possible spots in the Schedule, S, starting at index n.

Pseudo Code:

$Scheduler\{$

$\quad Let\ j_i = (p_i, j_i), where\ p_i\ is\ the\ penalty\ and\ d_i\ is\ the\ deadline\ of\ job\ i, 1\ \leq i \leq n$

$\quad Let\ J = \{j_1, j_2, \ldots j_n\}, for\ n\ jobs$

$\quad Let\ S \leftarrow \emptyset$

$\quad Let\ S(t)\ return\ an\ element\ in\ the\ set\ at\ position\ t, j_t$

$\quad k \leftarrow n$

$\quad mergesort(J)\ \#Sort\ in\ order\ of\ decreasing\ penalties\ p_1 \geq p_2 \ldots . \geq p_n$

$\quad for\ i \leftarrow 1\ to\ n\{$

$\quad\quad for\ t \leftarrow d_i\ to\ 1\{$

$\quad\quad\quad if\ S(t)\ is\ empty\{$

$\quad\quad\quad\quad S(t) = j$

$\quad\quad\quad\}else\{$

$\quad\quad\quad\quad S(k) = j$

$\quad\quad\quad\quad k -= 1$

3. Activity Selection Last to Start

In the greedy algorithm where criteria was to select an activity based on the first to finish, it was beneficial to examine the activity, a_i, from start to finish, $[s_i, f_i)$.

Instead of examining from start to finish, we can look from finish to start. Let $a'_i = (f_i, s_i]$, the interval is equal to a_i, but in reverse.

If we let A be a set of activities $A = \{[s_1, f_1), [s_2, f_2), \dots [s_n, f_n)\}$. We can observe the following, all choices are compatible, they do not overlap. Because each choice is compatible, the following must be true $f_1 \le s_1 \le f_2 \le s_2 \le \cdots f_{n-1} \le s_{n-1} \le f_n \le s_n$. The last activity in A, $[s_n, f_n)$ is the last to start in this set and is a valid first selection to A'. It follows that all the activities in A are a valid solution to A', but in reverse.

We can Prove by induction:

Let $P(n) \equiv \mathit{the\ activities\ in\ the\ solution\ A\ are\ also\ a\ valid\ in\ A', but\ in\ reverse.}$
That is $A = \{[s_1, f_1), [s_2, f_2) \dots . [s_n, f_n)\}\ \mathit{is\ compatible\ with\ A', where}$
$A' = \{(f_n, s_n], (f_{n-1}, s_{n-1}], \dots . (f_2, s_2], (f_1, s_1]\}$

Inductive Hypothesis: $P(n-1)\ \mathit{is\ true, then}\ P(n)\ \mathit{is\ also\ true.\ That\ is, if\ it\ is\ true\ that\ activites}, a_i, 1 \le i \le n-1\ , \mathit{in\ A\ are\ also\ in\ A', but\ in\ reverse, then\ it\ is\ true\ for}\ a_n\ \mathit{as\ well.}$

Base Case: P(1). An activity of one in $A = \{[s_1, f_1)\}$*, the activity is both the first to finish and the last to start, therefore,* $A' = \{(f_1, s_1]\}$ *. Thus P(1) is true.*

Inductive Step: Suppose activity, $a_k, k = n$*, is an activity that was chosen by the original greedy algorithm, then it must be true that* a_k *is compatible with* a_{n-1}*, that is*
$f_k \ge s_k \ge f_{n-1} \ge s_{n-1} \dots . \ge f_2 \ge s_2 \ge f_1 \ge s_1$.
(I.H.) Because it is true that P(n-1) is true, that is $a'_{n-1} = (f_{n-1}, s_{n-1}]$ *is currently the first selection in the set A', then activity* a_k*,* $\mathit{can\ be\ added\ before}\ a'_{n-1}\ \mathit{in\ the\ set\ A'}$*, because it is compatible.*
$a_k\ \mathit{satifies, the\ criteria\ of\ the\ new\ greedy\ algorithm, i.e\ it\ is\ the\ last\ to\ start\ and\ is\ is\ compatible\ with\ other\ acitiveis\ in\ the\ set\ A'.\ Therefore}\ P(n)\ \mathit{is\ true.\ QED.}$

4. Activity Selection PseudoCode

```
Activity{
  Let $A = \{a_1, a_2, \ldots a_n\}$ be the set of activites to be scheduled
  Let $a_i = (f_i, s_i), where\ f_i\ is\ the\ finish\ time\ and\ s_i\ of\ activity\ i, 1 \le i \le n$
  $Sort(A)\ in\ order\ of\ decreasing\ start\ times,$
  $S = \{a_1\}$
  $i = 1$

  $for\ j \leftarrow 2\ to\ n$:
    $if\ s_i \ge f_j$:
      $S = S \cup \{a_j\}$
      $i \leftarrow j$
     $A = A - \{a_j\}$
```

Assume mergesort will be used to sort A, then the running time of the sorting algorithm is

$$\theta(n \log n)$$

The for loop will iterate j = 2 to n, the comparison, which is runs in constant time, T(1)=1 will always execute during each iteration. The three statement may or may not execute. In the worst case the if statement and all the assignments execute, (n – 2) * 4. In the best case only the if statement execute and none of the assignment execute, (n – 2) * 1. Therefore we can say that the running time of the for loop is:

$$\theta(n)$$

Overall the running time is:

$$T(n) = \theta(n \log n) + \theta(n) \therefore T(n) = \theta(n\ logn)$$