Michael Smith

problem # 1

A) Algorithm:

1. Sort the job in terms of penalty in non-decreasing order
2. Sort the job in terms of deadline in non-decreasing order
3. For every job in te queue
4. Pick one whose penalty is minimum
5. See if its deadline is also min.
6. Run the job
7. End for loop
8. End

B)

* Algorithm is sorting based on the penalty and the deadline so it will take nlogn time each.
* After, we are assigning the job in O(n) so the time complexity is,

$$T(n) = O(n \log n)$$

Problem #2

* The Scheduler can exchange all start times with end times and can also reverse the time directions & it's logically the same problem. The greedy strategy of selecting the first activities is to end by selecting the last activity to start with. The same activities will be selected just in reverse order.

* Since there will be overlapping of the other activities, Institutions will not like activities that starts early in the Day. The activity will start at last and scheduled first. This helps the activities to stop overlapping.

* Assume that activities have been sorted by start times in decreasing order,

```
Schedule (int n, int start[1..n], int finish[1..n]) {

  Vector A = <1>          //Assume start[1..n] is sorted
    int previous = 1
      for i = 2 to n {
        if (A[i] S[previous]) {
          A = A + <i>
          previous = i
        }
      }
    return A
}
```

Problem #3:
Algorithm Description:

1. Sort activities by start times.
2. Get the first element and append to the selected activity(actSelected).
3. If the current activities finishing time is less than or equal to the start time of the previous activity that was selected, then append current activity to the selected activity variable(actSelected).

Pseudocode:
The theoretical running time would be O(nlogn),

```
lastToStart(s, f) {
        n = len(s)
        A = [a1]
        i = 1

        for m = 2 in n {
                if fm <= si {
                        A = AU[am]
                        i = m
                }
        }
        return A
}
```