Problem1:

a) I would have an array A to store the jobs in terms of penalty in increasing order. For each job, I would find a time slot i, such that slot is empty and i < deadline and i is greatest, then put the job in this slot and mark this slot filled.

b) Pseudocode:

```
JobSchedule(A)      // A is sorted array in increasing order
        Num = 0     // number of jobs
        While T is not empty
                Remove job i with smallest penalty
                If there's time slot j for i
                        Put job i in slot j
                Else
                        Num = Num + 1
                        Put job i in slot Num
```

The running time of this algorithm is O(nlogn).

Problem 2:

For this approach, each step we choose is the best option, and as we know that greedy algorithm is a technique which makes the best possible choice at the moment. We're trying to get the optimal solution by selecting the last activity to start that is compatible with all previously selected activities. Therefore, this is a greedy algorithm.

Problem 3:

Greedy approach:

1. Sort the activities in term of their finishing time in increasing order
2. Select the first activity form the sorted array
3. Do the following for the remaining activities in the sorted array
    a. If the start time of this activity is greater than or equal to the finish time of previous activity, then select this activity

Pseudocode:

```
LastToStart(Activities, n)
        mergeSort(Activities)
        Create an array A with size n       // assume the maximum number of of selected
                                            // activities can be n
        i ← 0
        A[0] ← Activities[0]
        For j from 1 to n
                If Activities[j].start >= Activities[i].finish
```

A[j] = Activities[i]

$i \leftarrow j$

Return A