1.
    a. Sort the jobs by ascending deadline. For loop i as time, if i is past deadline then add its penalty
    b. Pseudo

```
penalty = 0
arr = [[4,7], [1,5], [2,3], [1,10], [4,9]]
mergeSort(arr) # basic merge (nlogn), ascending deadline
for i in len(arr):
 if arr[i][0] <= i: # if deadline is below time, add penalty
   penalty += arr[i][1]
print(penalty)
```

    c. Merge sort + for loop = nlogn + n = O(nlogn)

2. Let $S_k = \{a_i \in S_k : S_i \geq f_k\}$ be the set of activities that end before activity a_k starts. Consider any non-empty subproblem S_k with activity a_m having the latest start time. Then a_m included in some maximum-size subset of mutually compatible activities of S_k.
   $f_m = min\{f_k : a_m \in S_{ij}\}$
   Then the following two conditions must hold.
    1. a_m is used in an optimal subset of S_{ij}
    2. S_{im} = Ø leaving S_{mj} as the only subproblem, meaning that the greedy solution produces an optimal solution.

   Consider any non-empty subproblem S_k with activity a_m having the latest start time. Then a_m included in some maximum-size subset of mutually compatible activities of S_k.
   Let A_k be an optimal solution for S_k and a_j be the activity in A_k with the latest start time.
   If a_j = a_m then the condition holds
   If $a_j \neq a_m$ then construct A_k' = A_k - {a_j} U {a_m}
   Since $f_m \geq f_j$ ⇒ A_k' is still optimal
   The activities in are disjointed since $f_m \geq f_j$. Since $|A_k|$=|A_k'|, we conclude that A_k is a maximum-size subset of mutually compatible activities for S_k and include a_m.

3. schedule.py
    a. Get ints from file into an array, pop and loop master array to segment into sets by set. Merge sort current working sets by ascending start time, reverse array. Loop through adding the job number of the set whose end time is <= to previous start time. Reverse output array, output.
    b. Pseudo

```
def main():
 arr = [] # master array of file ints
 with open('act.txt', 'r') as f: # get file as master array
   while True:
     line = f.readline() # get line from file
     if(not line): # check if line is eof
       break
     for i in line.split(): # get chars between space
       arr.append(int(i))
 iter = 1
 while(arr):
   N = arr.pop(0) # get number of sets
   sets = [] # array of sets, activity number, start time, end time
   for n in range(N):
     tmp = []
     for n in range(0, 3):
       tmp.append(arr.pop(0)) # get the three number of a set
     sets.append(tmp)
   mergeSort(sets) # basic merge (nlogn), ascending start time
   sets.reverse()

   ar = [] # get picked jobs
   ar.append(sets[0][0]) # take first
   num = sets[0][1] # start of last job
   for i in range(1, len(sets)):
     if(sets[i][2] <= num): # if deadline is less/equal to last start
       ar.append(sets[i][0]) # add to picked
       num = sets[i][1] # get new start
   sets = ar

   sets.reverse()
   print("Set", iter)
   print("Number of activities selected =", len(sets))
   print(sets, "\n")
   iter += 1
main()
```

c. Running time = Merge sort + for loop = nlogn + n = O(nlogn)