Solutions for Assignment 4

There are two ways to look at this problem:

1) Think backward in time. So loop over the events and convert every event with start and (e,f) to (-f,-t), now it's sorted by their finishing time! It's easy to see that if we have a solution for a one problem, it works for the one. In other words, if (s1,f1) doesn't conflict with (s2,f2), (-f1,-s1) doesn't conflict with (-f2,-s2) either.

```
Def revese_intervals(L)
      revesed_L=[] //empty list
      For i from len(L) downto 0:
            (s,f) = L[i]
            revesed_L.append((-f,-s))
      return revesed_L

Activitiy_Scheduling(L):
      L' = revese_intervals(L)
      soloution_reversed = FindSchedule(L')
      soloution = revese_intervals(Soloution_reversed)
      Return solloution
```

2) Another way to look at solution one, is to see we are actually instead of selecting the first activity to finish, we select the last activity to start that is compatible with previous selections. You should be able to replicate the prove by following the prove we discussed in the class.

2. We sort all the starting and finishing times and process from left to right. To keep track what event they belong to, and whether they are starting or finishng time, we label each timestamp with three attributes:

t.label = 'start' or 'end'
t.activity = the owner activity
T.room = the assigned room

```
def assign_room(activities):
    T=[] //empty list
    for a in activities:
        t = new timestamp()
        t.time=a.start
        t.label = 'start'
        t.activity=a
        T.append(t)
```

```
        t = new timestamp()
        t.time=a.end
        t.label = 'end'
        t.activity=a
        T.append(t)

T=sort T by t.time
For all (t,l,activity) in  T
    if t.label == 'start':
        if Q.empty():
            r = request_for_new_toom()
            t.activity.room = r
        else:
            r = Q.deque()
            T.activity.room = r
    if t.label == 'end':
        Q.enqueue(t.activity.room)
        h.activity.room = null
return Q.size()
```

The maximum number of rooms will be equal to the maximum number of conflicting events and that is easy to prove: when the algorithm asks for room *m*, it means that all the rooms are taken by the previous *m-1* events and no finish time has been seen since then, which means that the current even conflict with the previous events.
This problem is also called Interval Graph Coloring. Look this term up, and understand the relation between the two problems

3. Let's say the registration times are t1,t2,...tn. We argue that there is an optimal solution that starts from t1. Let's assume O is an optimal solution that does not start from t1, we shift the first employer to start right at t1 to t1+1, and shift all the subsequent possible conflicting employees until they don't overlap. It uses the same number of employees, therefore also optimal. So the solution simply is:
1- Let the first employer start from t1 to t1+1
2- Remove all the students in this range
3- Start from the first student not registered yet

4. a) True, is a<b, obviously $a^2<b^2$, so squaring doesn't change the order (which is all Kruscal cares about)

b) True: Suppose T and T' both be MST for G and T has at least one edge e = (u,v) not in T' . Add e to T' and there will be a cycle. Now find the most expensive edge in this cycle and remove it, there are two cases:
Case 1. The most expensive edge is not e, so now you have T'' less expensive than T', which is a contradiction

Case 2. e is the most expensive node in T'. Follow the path from e to w in T' and bring one which is not already in T (there should be one, other wise T would have a cycle), and replace it with e, and again, you would end up with a tree less expensive than T, which is a contradiction

5. a) Similar to last question, add the new edge to the tree, and check if the new edge is the most expensive edge in the cycle. If yes, T is still the MST. Finding a cycle can be done in |V| (you did it in the last assignment), and finding the minimum edge also can be done in |V|, be cause there cannot be more than |V| edges in the new tree (it was a tree).

b) Exactly like the previous step, only remove the the most expensive edge in the cycle and obtain T'. A formal proof can be lengthy (and is not asked for here), but you can try to show why this is the new MST by showing that if any edge e' is missing from T' to be and MST, e' should've been in T from the beginning, hence a contradiction. The new algorithm has the same O(|V|) complexity.