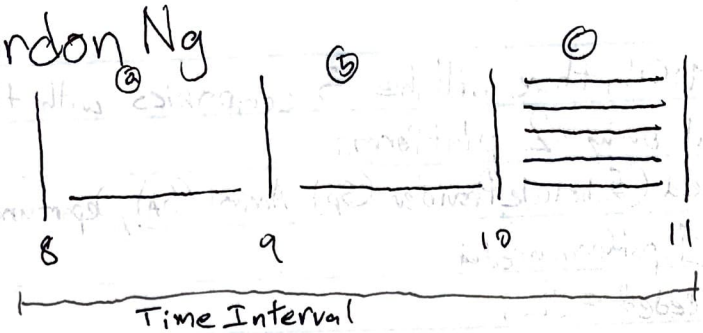
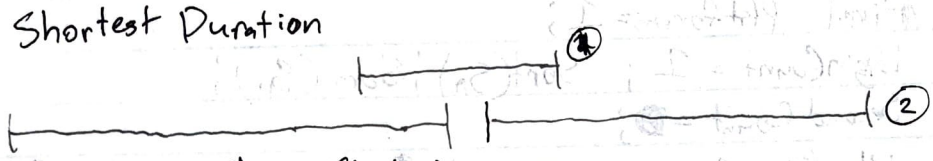
Gordon Ng

a)



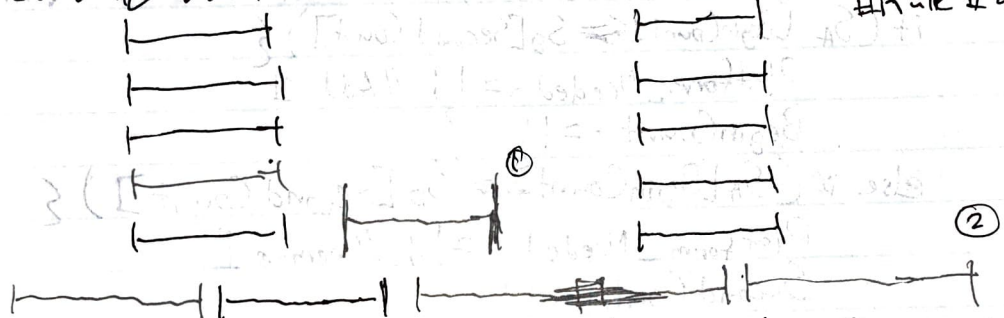
You can make 5 different solutions by adding a+b and choosing a new c.

b) Shortest Duration



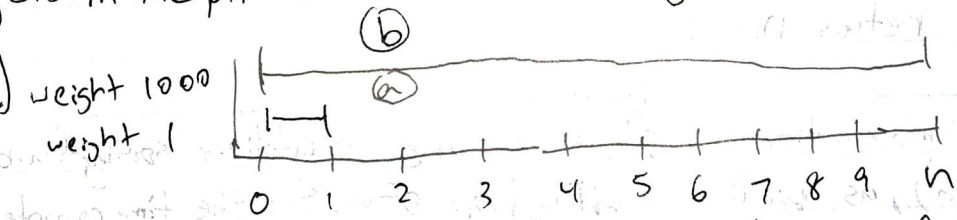
Fewest [redacted] conflicts:

Rule #4 in lecture



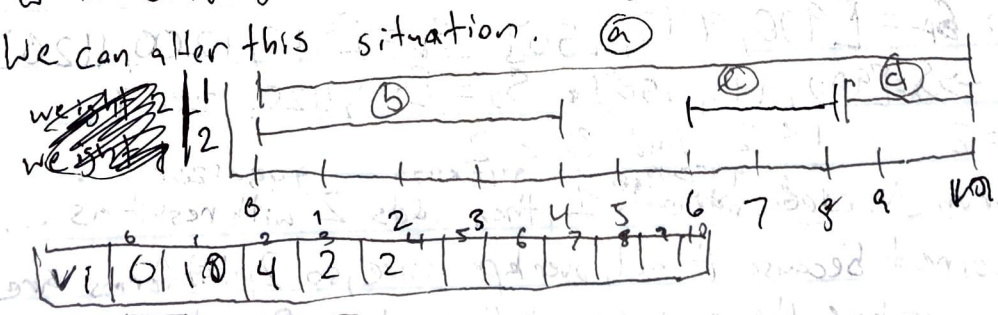
Shortest Duration is not optimal because it will select figure 1 which then blocks the whole schedule for the figure 2 classes
Fewest Conflicts is not optimal because it will select figure 1 which gets in the path of the schedule in figure 2.

c)



With Unweighted schedule this greedy algorithm fails.

We can alter this situation.



vi	0	10	4	2	2						
pi	0	0	b	c+b							

This algorithm would choose b + c + d as the optimal solution but the total maximal value of any compatible solution would be @ (larger than predicted)

2a) No at exactly 9:01, there will be 3 companies with the proposed time slot but only 2 platforms.

b) FewestPlatformNeeded (ScheduleProvider (S_p), Arrival (S_A), Departure (S_D))

```
// initialize 1 platform needed
Platform_Needed = 1;
Final_Platforms = 1;
BeginCount = 1; Sort(S_A); Sort(S_D);
SecondCount = 0;
while (BeginCount < len(S_Arrival) And SecondCount < len(S_Arrival))
if (S_A [BeginCount] <= S_D[SecondCount]) {
      Platform_Needed += 1; // Add 1
      BeginCount += 1; }
else if (S_A[BeginCount] > S_D[SecondCount]) {
      Platform_Needed -= 1; // remove 1
      SecondCount += 1;
  }
  if (Platform_Needed > result) {
      result = Platform_Needed;
  }
  return result
}
```

This algorithm's runtime is O(nlogn) because it utilizes sorting function of O(nlogn), as well as a while loop of n. So, the time complexity is O(nlogn)

Correctness: S_A = [900, 945, 950] S_D = [910, 1200, 1120]

First sort → S_A [900, 945, 950] S_D = [910, 1120, 1200]

Len S_A = 2 1 < 2 And 0 < 2

945 ≥ 910 945 ≤ 1120 945 < 1200

Platform_Needed, removes 1, then adds 2 with result as 2.

This is correct because it only overlaps once, so 2 platforms are needed. You must have the same amount of Arrivals as Departures, so this would work for all cases.

You start with 1 platform initialized, then you loop through arrivals and departures because every arrival is mapped to a departure. Ex: If a train comes, it has to leave. Adds a station if overlapping, removes a platform if it is not needed / can be scheduled.

2c. Few Schedule ($S_{provider}$, $S_{Arrivals}$, $S_{Departures}$) {

```
Platform_Needed = 1;
Final_Platforms = 1;
BeginCount = 1;
SecondCount = 0;
Sort(S_A);
Sort(S_B);
while(BeginCount < len(S_Arrival) AND SecondCount < len(S_Arrival))
    if (S_A[BeginCount] <= S_D[SecondCount]) {
        Platform_Needed++;
        Final_Platforms ++; }
    else if(S_A[BeginCount] > S_D[SecondCount]) {
        Platform_Needed --;
        SecondCount ++;
    }
    if (Platform_Needed > result) {
        result = Platform_Needed;
    } d = {} <- dictionary
for (i=0, i=results+1 (Example: 0,1,2,3,results), i++) {
    d['Platform' i] = None (create a dictionary)
comment //## {Plat1 : None, Plat2 : None, Plat3 : None ...}
while (i=0, i = range(len(Sproviders)), i++) {
    if Sproviders[i] == 'Acela'
        d[Platform1'] = [Sprovider[i] SArrival[i], Sdeparture[i]]
    else:
        d.sort() //# sort all other trains by train length into other
schedule = d //# dictionary //#platforms in d dictionary.
return result, schedule //# if there are overlaps, start by sorting into
                        //# platform 1, then use platform 2, and so on.
                        //# since we calculated the minimum # of platforms,
                        //# there should always be an available platform.
```

This function is built from 26) where we sort the schedule, merge in a loop fashion $dep and $Arrival to figure out overlaps and non-overlaps then adds for overlaps, removes for non-overlaps, to get the minimum amount of platforms. Once we get the min amount, we initialize a dictionary and assign keys as the Platforms and immediately take Acela's schedule and place it in the dictionary because it will always be Platform 1. Then we run a sorting function, similar to what we have above, just alternating between stations when one Platforms is full during their schedule by comparing $Arrival and $departure times.

The Bigger the train the more priority?

proof of correctness:

If we have $Prowder = [Acela, Amtrak, MBTA]

$Arrival = [900, ~~1200~~, 945, ~~950~~]

$departure = [910, ~~945~~, ~~950~~, 1200, 1120]

We get: result = 2

schedule = { Platform 1: [Acela, 900, 910], [Amtrak, 945, 1200]
Platform 2: [MBTA, 950, 1120] }