HW3 ANSWER

Q1-)

Öncelikle belirtmek isterim ki film çok güzel, o devir araba yapmak gibi bir işe kalkışmaları muazzam Cemal Paşanın azmi ve isteği zaten tartışılamaz, ancak yine her zaman ki gibi başarılsa dahi sonunun böyle olması üzücü, kendi kendimizin düşmanıymışız gibi araba yapılmış olsa dahi basının direkt olarak kötü olarak manşet atıp DEVRİM i lekelemesi başlamadan direkt kaybettiğimizi gösteriyor. Zamanın kısıtlı şeyleri ile bir şeyler yapılmaya kalkışılmış ve bence başarılı da olunmuş keşke önü kesilmeseymiş de devamı gelseymiş ve şuan kendi arabamızı üretebiliyor olsaydık. Devrimin tasarımı özellikle çok hoşuma gitti o zamanda press makineleri dahi yokken elimizde kendileri başka bir makine ile o çelikleri düzeltmeyi akıl etmelerini çok beğendim. Filmin oyuncu kadrosu da çok güzel ve çok başarılı oyunculuk sergilenmiş.

Sonuç olarak, keşke üretilebilseymiş o zaman, şuan ülke olarak eminim ki çok farklı yerlerde olurduk. Bu ülkenin kaybettiği çok şey var, dışarının etkilerinden daha çok kendi milletinden dolayı kaybettiği çok şey var. Filmde de geçtiği gibi “Türkiye'de hiçbir başarı cezasız kalmaz.” Umarım ilerleyen senelerde çok daha iyi yerlere geliriz.

Q2-)

Verilen öğretmen sayısı kadar tüm durumların permütasyonlarını bir listeye, listeler halinde üretip ekledim. Bulduğum tüm permütasyonları her birini gezip hesaplayarak minimum süre olanı bulup return ettim.

```
    x=len(inpTable)
    z=len(inpTable[0])
    arrays=allPermList(x,z)
    min=findValue(arrays[0],inpTable)
    tmin=0
    returnList =[]
    tempList = []
    for i in range(len(arrays)):
        tempList = arrays[i]
        tmin=findValue(tempList,inpTable)
        if tmin<=min:
            returnList = tempList
            min=tmin
    #print(returnList,"# RA0,...")
    return returnList,min
```

Arrays te tüm permütasyonlar bulunmakta, dolayısı ile complexity analysis için verilen input kadar yazdığım kod her seferinde kesin öncelikle tüm permütasyonları bulacak. Bu durumda şunu ifade ediyor: (n!) kesinlikle permütasyonları bulmak için çalışacak. Yazdığım diğer helper fonksiyonlarının complexitysinin, bu ifadenin yanında pek bir anlamı yok. Bundan dolayı, worst case,best case ve avarage case durumlarında da yazığım program O(n!) complexity de çalışacak. NOT= Yazdığım program sadece kare matrislerde çalışmakta hocam.

Q3-)

BFS algoritmasından yardım alarak implement ettim, minimum maliyet hesabını. Verilen graf taki tüm vertexleri geziyorum ve connected olanları BFS ile return edip, bir set e atıyorum. Graf içindeki tüm connected olan grafları set e topladıktan sonra oradaki set üzerinden maliyet hesabını yapıyorum. Set teki eleman sayısı kadar laboratuvar kuruyorum, set teki her elemanın eleman sayısının bir eksiği kadar da yol vardır(set teki her elemanda). Kurulması gereken tüm yolları ve kurulması gereken tüm laboratuvarları bulduktan sonra maliyet hesabını yapıyorum.

Worst-Case Analysis:

```
lst = []
for i in range(1,len(graphGTU)+1):
    s=gtuBFS(graphGTU,i)
    all.add(tuple(s))
labCount = len(all)
lst = list(all)
for j in range(len(lst)):
    roadCount = roadCount + (len(lst[j])-1)
minCost = l*labCount + r*roadCount
```

findMinimumCostToLabifyGTU fonksiyonunda verilen graf ta kaç tane ayrı ayrı connected graf var ise o kadar dönüyor içindeki for. Dolayısı ile complexitiy me etkisi kaç kere gönüyorsa o dur. Verilen graf taki her vertex ayrı bir graf temsil etmiş olsa n kez çalışacağını söylesek çağırılacak olan BFS fonksiyonunun da complexity si vertex sayısı + edge sayısı olacağından her biri de ayrı olduğundan pek dönmeyecek BFS fonksiyonundaki for dolayısı ile worst-case durumu (BİG-O) O(V+E) olur. Birde eğer verilen graf elemanları hepsi bağlı ise yani verilen grafın tamamı tek bir connected graf ise o zamanda yazdığım minimumcost fonksiyonundaki for tek kez dönecek ama bu seferde BFS fonksiyonunun içindeki for tüm vertex leri gezeceğinden dolayı tekrarda complexity si (BİG-O) O(V+E) olacağından, genel olarak baktığımızda complexity si O(V+E) olacaktır.

Q4-)

12,34,54,2,3 for insertion sort.

12 34 54 2 3
12 34 54 54 3
12 34 34 54 3
12 12 34 54 3
2 12 34 54 3
2 12 34 54 54
2 12 34 34 54
2 12 12 34 54
2 3 12 34 54

12,34,54,2,3 for Shell sort. (first gap=n/2)

12 34
54 2
3

3 2
12 34
54

3 2 12 34 54

İnsertion sort uygularsak son aşamada.

3 2 12 34 54
3 3 12 34 54
2 3 12 34 54

Insertion sort arayin büyüklüğüne bakmaksızın eğer dizi elemanları sıralıya yakınsa hızlı algoritmadır. Shell sort dediğimiz sort algoritması ise insertion sortların belli aralıklarla tekrarlarından oluşmaktadır. İnsertion sort her elemanı dizinin diğer tüm elemanları işe karşılaştırır, Shell sort ise birbirinden belirli uzaklıktaki elemanları karşılaştırır, tüm elemanlar sıralı olana kadar, belirli aralık kapanana kadar bu işlemi yapar. Baktığımızda eğer elemanlar birbirine çok sıralıya yakın değilse Shell sort daha etkilidir ancak sıralıya yakınsa insertion sort algoritması daha etkili olacaktır.