1. **Planificarea proiectelor cu profit maxim (varianta cu interval de desfășurare) – complexitate $O(n^2)$, unde n reprezintă numărul proiectelor**

```
# functie comparator pentru a sorta proiectele crescator dupa data de sfarsit
def cmpProiecte(t):
    return t[2]

f = open("proiecte.in")

# lista proiectelor in care am adaugat un prim proiect "imaginar"
# pentru a avea proiectele indexate de la 1
lp = [("", 0, 0, 0)]
for linie in f:
    # un proiect = un tuplu (denumire, data inceput, data sfarsit, profit)
    aux = linie.split()
    lp.append((aux[0], int(aux[1]), int(aux[2]), int(aux[3])))

f.close()

# n = numarul proiectelor (scadem 1 pentru primul proiect "imaginar")
n = len(lp) - 1

# sortam proiectele crescator dupa data de sfarsit
lp.sort(key=cmpProiecte)

# pmax[i] = profitul maxim care se poate obtine din primele i proiecte
# ult[i] = cel mai mare indice j (1 ≤ j < i) al unui proiect care nu se
suprapune cu proiectul i
# sau 0 dacă nu există nici un astfel de proiect
pmax = [0] * (n + 1)
ult = [0] * (n + 1)

for i in range(1, n+1):
    for j in range(i-1, 0, -1):
        if lp[j][2] <= lp[i][1]:
            ult[i] = j
            break

    if lp[i][3] + pmax[ult[i]] > pmax[i-1]:
        pmax[i] = lp[i][3] + pmax[ult[i]]
    else:
        pmax[i] = pmax[i-1]

fout = open("proiecte.out", "w")

# reconstituirea unei solutii
i = n
while i >= 1:
    if pmax[i] != pmax[i-1]:
        fout.write(lp[i][0] + " " + str(lp[i][1]) + " " + str(lp[i][2]) + " "
+ str(lp[i][3]) + "\n")
        i = ult[i]
    else:
        i -= 1

fout.write("\nBonusul echipei: " + str(pmax[n]))

fout.close()
```

2. Sortarea topologică a unui graf orientat (cu n vârfuri și m arce) – complexitate O(n+m)

```
# Fiserul text graf.txt contine pe prima linie numarul de varfuri n,
# iar pe urmatoarele linii arcele grafului

f = open("graf.txt")

n = int(f.readline())

# numarul de predecesori ai fiecarui varf 1,2,...,n
nrp = [0] * (n + 1)

# lista succesorilor fiecarui varf
succ = {i: [] for i in range(1, n+1)}

# pentru fiecare arc (x,y) din graf adaugam varful y in lista succesorilor
# lui x si crestem numarul predecesorilor lui y cu 1
for lin in f:
    aux = lin.split()
    x = int(aux[0])
    y = int(aux[1])
    succ[x].append(y)
    nrp[y] += 1

f.close()

# q este coada in care pastram varfurile care nu au predecesori
q = []
for v in succ:
    if nrp[v] == 0:
        q.append(v)

# sol este o lista in care construim sortarea topologica a grafului
sol = []
while q != []:
    # extragem din q elementul curent (un varf care nu are predecesori)
    # si il adaugam in sortarea topologica
    v = q.pop(0)
    sol.append(v)

    # pentru fiecare succesor al varfului curent
    # actualizam numarul de predecesori si, eventual, coada q
    for s in succ[v]:
        nrp[s] -= 1
        if nrp[s] == 0:
            q.append(s)

if len(sol) != n:
    print("Graful contine cicluri!")
else:
    print("O sortare topologica a grafului:", sol)
```

3. Determinarea drumurilor de lungime minimă într-un graf orientat aciclic

https://en.wikipedia.org/wiki/Topological_sorting#Application_to_shortest_path_finding
https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/