Lăzăroiu Teodora-Bianca, 141

1. a)
```
def apartine(mult, *liste):
    d = {}  nr = 0
    for x in mult:
        for listă in liste:
            if x in listă:

                d.update({x : (nr, listă.count(x))})
```

b)
```
perechi = [(a,b) for a in range(1,11) for b in range(1,11)]
```

c)
```
def f(v,p,u):
    if u == p:
        return v[u]
    else:
        m = (p+u) // 2
        for i in range(p, m+1):
            v[i] = v[i] + v[u-i+p]
        return f(v,p,m)
```

$f(L, 0, n-1)$ $\qquad T(1) = 1$

$T(n) = T(n/2) + n/2 = T(n/4) + n/4 + n/2 =$

$= T(n/8) + n/8 + n/4 + n/2 = \dots =$

$= T(1) + \dots + n/8 + n/4 + n/2 =$

$= n/2 \cdot \frac{(\frac{1}{2})^{\log_2 n} - 1}{\frac{1}{2} - 1} = n/2 \cdot \frac{(\frac{1}{2})^{\log_2 n} - 1}{-\frac{1}{2}} =$

$= n \cdot (\frac{1}{2})^{\log_2 n} - 1 = n \Rightarrow O(n)$ complexitate

Lăzăroiu Teodora - Bianca, 141

2. Greedy

```
g = [int(x) for x in input().split()] # capacitățile
c = int(input)  # capacitate bazin      # găleților
g.sort(reverse = True) # descrescător după capacita
c_curent = 0 # capacitatea curentă
ls = []
for x in g:
    if c_curent + x <= c:  # dacă o găleată mai
        ls.append(x)       # are loc în bazin
        c_curent = c_curent + x

if c_curent != c: # dacă bazinul nu este plin
    print("nu se poate")
else:                 # dacă se poate afișează listă cu
    print(*ls)        # capacitățile găleților
```

Algoritmul se încadrează în metoda Greedy deoarece sunt alese întâi valorile cele mai mari pentru umplerea bazinului

Corectitudinea este dată de datele problemei care impun ca o găleată să aibă capacitatea strict mai mare decât suma găleților mai mici

Algoritmul are complexitate $O(n \log_2 n)$ deoarece este folosit sort cu complexitatea $O(n \log n)$ iar for-ul are doar complexitatea n

Lăzăroiu Teodora - Bianca, 141

4. Backtracking

a)
```
def back(k):
    global m
    global d ; global ok
    if k == m+d+1: # nr. total de elemente e d+m
        print(* x[1:], sep=", ") ok = 1
    else:
        for i in range(1, d+m+1): # ia valori de la
            x[k] = i                    # i la d+m
            if x[k] not in x[:k]:
                * if x[k] > d and x[k-1] <= d: # dacă
                    back(k+1)  # elementul curent e m
ok = 0                         # elem. anterior trb să
d, m = [int(x) for x in input().split()] # fie d
x = [0 for i in range(d+m+1)]
back(1)
if ok == 0: # dacă nu s-a afișat nimic
    print("nu există modalități de aranjare")
```

b) la instrucțiunea *:

```
if x[k] > d and x[k-1] <= d and x[1] <= d
    and x[d+m] <= d:
```

Se adaugă condiția că primul și ultimul el. să aparțină intervalului [1, d]

a) Programul adaugă la partea de verificare condiția ca dacă elementul curent aparține interv. [d+1, d+m] (este de murături) elementul anterior trebuie să aparțin lui [1, d] (de dulceață)