TAP C1_handouts: Introducere

1. **SUBSET SUM**
   Să se verifice dacă se poate alege dintr-un vector de numere întregi o submulțime care să aibă suma elementelor 0.

   Pentru -2, 1, -1, -5, 8, 2 se poate alege submulțimea -2, -1, -5, 8.
   Pentru -2, 6, -3, -5, 11 nu se poate alege o submulțime astfel încât suma elementelor să fie 0.

- Problemă echivalentă: Există o submulțime care să aibă suma elementelor egală cu ***s*****?** Care sunt valorile *s* pentru care putem răspunde NU după o singură parcurgere a vectorului?
- Propuneți un algoritm. Ce complexitate are algoritmul propus?

2. **TEMA 3 SUM**
   Să se verifice dacă se poate alege dintr-un vector de numere întregi o submulțime cu exact 3 elemente care să aibă suma elementelor 0.

3. **LOAD BALANCE**
   Se consideră *n* task-uri, pentru finalizarea task-ului *i* fiind necesare t[i] unități de timp. Având la dispoziție *m* resurse, care pot finaliza task-urile lucrând în paralel și având la dispoziție timp nelimitat, să se atribuie fiecare task uneia dintre resurse (a[j] = i task-ul *j* va fi alocat resursei *i*) astfel încât timpul utilizat pentru finalizarea task-urilor să fie minim, i.e să se minimizeze:

$$makespan = \max_i \sum_{A[j]=i} T[j]$$

- Care este complexitatea algoritmului Greedy propus? Se poate îmbunătăți?

```
#input
#7 3
#10 4 5 8 9 2 6
#output
# [0, 1, 2, 1, 2, 0, 0]
# [18, 12, 14]

line = input().split()
n, m = int(line[0]), int(line[1])
#t[i] time task i
t = [None] * n
```

```
#r[i] total time, resource i
r = [0] * m

#a[i] resource performing task i
a = [None] * n

line = input().split()
for i in range(n):
    t[i] = int (line[i])

for i in range(n):
    min_r = min(r)
    min_i = r.index(min_r)
    a[i] = min_i
    r[min_i] += t[i]

print(a)
print(r)
```

```
from heapq import *

line = input().split()
n, m = int(line[0]), int(line[1])
t = [None] * n
r = [None] * m
for i in range(m):
    r[i] = [0, i]
heapify(r)

a = [None] * n

line = input().split()
for i in range(n):
    t[i] = int (line[i])

for i in range(n):
    res = heappop(r)
    a[i] = res[1]
    heappush(r, [res[0] + t[i], res[1]])

print(a)
print(r)
```

4. **CORECTITUDINEA ALGORITMILOR**

Propuneți invarianți pentru a demonstra corectitudinea următorilor algoritmi:

```
a) v = [4, 3, 2, 5, 14, 7, 9 ,8 , 12]
max = v[0]
min = v[0]
i = 0
while i < len(v):
    if max > v[i]:
        max = v[i]
    if min < v[i]:
        min = v[i]
    i += 1
print(min, max)
```

```
b) x = c
y = 0
while x > 0:
    x -= 1
    y += 1
print (x, y)
```

```
c) v = [4, 3, 2, 5, 14, 7, 9 ,8 , 12]
n = len(v)
for i in range(n):
    min = v[i]
    pozmin = i
    for j in range(i+1, n):
        if v[j] < min:
            pozmin = j
            min = v[j]
    v[i], v[pozmin] = v[pozmin],v[i]
print(v)
```

```
d) x = 105
y = 84
u = x
v = y
while (x != y):
    if x > y:
        x = x - y
        u = u + v
    else:
        y = y - x
        v = u + v
print(x, (u + v)//2)
```