Fundamentele programării

1 Scrieți o funcție care sortează o lista de numere folosind: mergsort. Subiect eliminatoriu. (1p)

2 Specificați si testați următoarea funcție (2p):

```
def f2(n):
    if n<=0: raise ValueError()
    l = [x for x in range(n-1,-1,-1)]
    for i in range(n-1):
        l[i+1] += l[i]
    return l[-1]
```

3 Analizați complexitatea timp si spațiu a următorului algoritm. (2p).

```
def f(l):
    if len(l) == 1:
        return l[0]
    if l[0]==0:
        return 0
    return l[0] * f(l[1:])
```

4 Folosind metoda Divide et impera scrieți o funcție pură care calculează numărul de numere negative într-o lista de numere. Datele trebuie împărțite in 2 parți egale la fiecare pas. (2p).

5 Generați toate sublistele unei liste date, cu proprietatea ca sublistele conțin ori doar numere pare ori doar numere impare. Descrieți schematic soluția (candidat, consistent, soluție) bazată pe metoda Backtracking (fără implementare) (2p)

1.
```
def mergesort(l, start, end):
    if end - start <= 1:
        return
    mid = (end + start) // 2
    return mergesort(l, start, mid)
    return mergesort(l, mid, end)
    merge(l, start, mid, end)

def merge(l, start, mid, end)
    i = j = 0 ; K = start
    left = l[start: mid]
    right = l[mid: end]
    while i < len(left) and j < len(right).
        if left[i] < righ[j]:
            l[K] = left[i]
            i += 1
        else:
            l[K] = right[j]
            j += 1
```

```
        k += 1
    while i < len(left):
        l[k] = left[i]
        i += 1
        k += 1
    while j < len(right):
        l[k] = right[j]
        j += 1
        k += 1
```

2. """ Funcția realizează suma parțială a elementelor [n-1, ..., 0], adică face suma elementelor din listă.
:param n: lungimea listei, iar n-1 este primul element din listă
:type n: int
:return: suma
:raises: ValueError dacă n este negativ sau 0
"""

```
def test_f():
    n = 5
```

```
assert f2(m) == 15
n = 1
assert f2(n) == 1
n = -5
try:
    f2(n)
    assert False
except ValueError:
    assert True

test_f1()
```

3. TIME COMPLEXITY

BC: pe prima poziție a listei este elementul 0 =>

=> $\Theta(1)$

WC: nu există 0 în listă => $T(n) = T(n-1) + 1 \Rightarrow \Theta(n)$

AC: $\sum \overbrace{P(i)}^{\text{probabilitatea să fie pe poz}} \overbrace{E(i)}^{\text{poziția o întoarce}} = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \dots n + \frac{1}{n} = \frac{1+2+\dots+n}{n} = \frac{n(n+1)}{n} = \frac{n^2+n}{n}$

$= \frac{n(n+1)}{n} = n+1 \in O(n)$

SPACE COMPLEXITY

$$T(n) = \begin{cases} 1, & n = len(l) = 1 \\ T(n-1) + n, & \text{otherwise where } n \text{ is the space time complexity of slicing} \end{cases}$$

$T(n) = T(n-1) + n$

$T(n-1) = T(n-2) + n - 1$

$T(1) = 1$

$T(n) = n + (n-1) + (n-2) + \dots + 1 = \frac{n(n+1)}{2} = \frac{n^2+n}{2} \in O(n^2)$

4.
```
def f(l):
    if len(l) == 1:
        if l[0] < 0:
            return 1
        return 0
    m = len(l) // 2
    return f(l[:m]) + f(l[m:])
```

5. Spatial de cautare

lista = $(x_1, \dots, x_n)$

$S = \{x_1, \dots, x_n\}$

Candidat

$x = (x_0, \ldots, x_k)$

$x_i \in$ lista

Condiție consistent

$x = (x_0, \ldots, x_k)$ consistent dacă $1 \leq k \leq$ lungime(listă) și $\forall i, j \in \{0, \ldots, k\}$, $x_i \% 2 == x_j \% 2$

Condiție soluție

$x = (x_0, \ldots, x_k)$ soluție dacă x consistent și $k \geq 1$