Fundamentele programării

1 Scrieți o funcție care implementează sortarea prin selecție. Subiect eliminatoriu. (1p)

2 Specificați si testați următoarea funcție (2p):

```
def f(n):
    if n<0: raise ValueError()
    if n <= 1: return n
    l = [0] * (n + 1)
    l[1] = 1
    for i in range(2, n+1):
        l[i] = l[i - 1] + l[i - 2]
    return l[n]
```

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

```
def f(n):
    s = 0
    for i in range(n*n):
        for j in range(i):
            k = 1
            while k < n:
                k *= 2
                s += 1
```

4 Folosind metoda Divide et impera scrieți o funcție pură care găsește maximul într-o lista de numere date. Datele trebuie împărțite in 2 parți egale la fiecare pas. Ex. L = [2,8,4,1,4,5,6,7] returnează 8 (2p).

5 Pentru un n dat generați toate secvențele de paranteze si acolade care se închid corect. Exemplu: n=4 → 8 soluții: (()), ()(), (){}, ({}), {()}, {{}}, {}(), {}{}. Descrieți schematic soluția (candidat, consistent, soluție) bazată pe metoda Backtracking (fără implementare) (2p)

1.
```
def f(lista):
    for i in range(0, len(lista)):
        for j in range(i+1, len(lista)):
            if lista[i] > lista[j]:
                lista[i], lista[j] = lista[j], lista[i]
```

2.
```
"""
Funcția generează șirul lui Fibonacci: 0,1,1,2... până la al n-lea termen și îl returnează.
:param n: reprezintă indicele ultimului element generat
:type n: int
:return: al n-lea termen generat
:type return: int
"""
def test_f():
    try:
        f(-1)
        assert False
    except ValueError:
        assert True
    assert f(0) == 0
```

```
assert f(1) == 1
assert f(2) == 1
assert f(4) == 3
test_f()
```

3. Time Complexity:

$BC = WC = AC$

$$T(n) = \sum_{i=1}^{n^2-1} \sum_{j=1}^{i-1} \log_2 n = 0 \cdot \log_2 n + 1 \cdot \log_2 n + \dots + (n^2-1) \cdot \log_2 n = (1+2+\dots+n^2-1) \log_2 n =$$

$$= \frac{n^2(n^2-1)}{2} \log_2 n \in \Theta(n^4 \log_2 n)$$

Space Complexity:

$\Theta(1)$, deoarece avem un număr constant de variabile și nu sunt create structuri precum liste.

4.
```
def f(l):
    if (len(l)==1):
        return l[0]
    mid = len(l)//2
    a = f(l[:mid])
    b = f(l[mid:])
    if a > b:
        return a
```

return b

5 Spațiu de căutare

$S = \{'(', ')', '\{', '\}'\}$

Candidat

$x = (x_0, \dots, x_k)$, $x_i \in S$, $1 \leq k \leq n$

Consistent

$x = (x_0, \dots, x_k)$ consistent dacă $\forall x_i \in (x_0, \dots, x_{k/2})$ $\exists x_{k-i+1} \in (x_{k/2+1}, \dots, x_k)$ a.î. $x_i = f(x_{n-i+1})$, unde $f^{-1}: S \to S$, $f^{-1}('(') = ')'$, $f^{-1}(')') = '('$, $f^{-1}('\{') = '\}'$, $f^{-1}('\}') = '\{'$, $k \leq n$.

Soluție

$x = (x_0, \dots, x_k)$ soluție dacă $k = n$ și x consistent