

Fundamentele programării

R2

1. Given the following code fragment what is the result of the execution: **(1p)**

<pre> def f(a, b, c): a = a + 1 b.append(3) c = c + [3] a = 7 b = [1, 2] c = [1, 2] f(a, b, c) print(a, b, c) </pre> <p>var. local</p> <p>$[1, 2]$ $[1, 2]$</p>	a) print: 7 [1, 2] [1, 2]
	b) print: 8 [1, 2, 3] [1, 2, 3]
	c) print: 7 [1, 2, 3] [1, 2]
	d) error on line: c = c + [3]

2. Please specify and test the following function: **(2p)**

<pre> def f(l): if l == None: raise ValueError() for e in l: if e % 2 == 1: return True return False </pre>	<p>'''Funcția verifică dacă lista l conține numere impare. : param l: lista în care căutăm : type l: list : return: bool : raises: ValueError dacă lista e None.'''</p>	<pre> def test_f(): l = [1, 2] assert f(l) == True l = [2] assert f(l) == False l = None try: f(l) except ValueError: assert True test_f() </pre>
---	---	---

3. Asymptotic analysis of the time complexity (best case, average case, worst case). Please also indicate the extra-space complexity. **(2p)**

<pre> def f(n): # n - integer number s = 0 m = n while m != 0: s = s + m / 10 m = m / 10 return s </pre>	<p>BC = WC = $\log_{10} n$ Extra space complexity: $\sim(1)$</p>	<pre> def a(n): s = 0 for i in range(1, n + 1): m = 2 * n + 1 s = s + f(m) return s </pre>	<p>BC = WC : numărul de pași făcuți de for: $\sum_{i=1}^n f(m)$ $f(m) = \log_2(2n+1)$ pași $\in \Theta(\log n)$ $\Rightarrow T(n) = O(n \log n)$ Extra space complexity: $\Theta(1)$</p>
--	--	--	---

4. Let us consider a list a_1, a_2, \dots, a_n of integer numbers. Using the “Divide et Impera” programming method, write, specify and test a function to compute the number of even elements from the list. **(2p)**

5. For the following problem, please indicate the most **APPROPRIATE** programming method (*Backtracking, Divide et Impera, Greedy, Dynamic Programming*) that can be used for solving it. Please justify the method's applicability and analyse the problem solving according to the particularity of the selected programming technique (*without implementation*). **(2p)**

Give all the possibilities to decompose a given natural number n as a sum of prime numbers
(**Example:** For $n=15$, the solutions are 3, 5, 7 and 2, 13)

4. def f(l: list) → int:
+
funcția realizează suma numerelor pare din listă
: param l: lista în care căutăm numerele pare
+
: return: suma numerelor pare

if l == [] or l == None:
+
return 0

if len(l) == 1:
+
return 0

if l[0] % 2 == 0:
+
return l[0]
+
return 0

lungime = len(l) // 2

+
return f(l[lungime]) + f(l[:lungime])

def test_f():

l = [1, 2, 3]

+
assert f(l) == 2

l = []

+
assert f(l) == 0

+
test_f()

5. O alegere bună pentru a rezolva problema este metoda backtracking deoarece putem adăuga într-o listă toate numerele prime pentru un n dat (civruul lui Eratosthenes), iar apoi folosind backtrackingul vom putea genera soluții ale cărei sumă este egală cu n .

Spatiu de căutare

$$S = \{x \leq n, x \text{ este număr prim}\}$$

Candidat

$$C = (x_1, \dots, x_k), \sum_{i=1}^k x_i < n$$

Soluție

$$S = (x_1, \dots, x_k), \sum_{i=1}^k x_i = n$$