

Fundamentele programării

1 Implementați căutare binară recursiv și nerecursiv. Subiect eliminatoriu. (1p)

2 Specificați și testați următoarea funcție (2p):

```
def f(n):  
    if (n<=0): raise ValueError()  
    while n>0:  
        c = n % 10  
        n = n//10  
        if (c%2==0): return True  
    return False
```

3 Analizați complexitatea timp și spațiu a următorului algoritmului. (2p).

```
def s(l,poz=0):  
    if poz<len(l):  
        a = s(l,poz+1)  
        b = s(l,poz+1)  
        return a + b  
    else: return 1
```

4 Folosind metoda Divide et impera scrieți o funcție pură care verifică dacă o listă de numere conține cel puțin un număr par. Datele trebuie împărțite în 2 părți egale la fiecare pas. Ex: [1,3,4,5] -> Adevărat (2p).

5 Se dă o listă de numere. Găsiți sublista cea mai lungă cu numere prime crescătoare. Folosiți programare dinamică, se cere: recurența și implementare iterativă în Python. Ex: Pentru lista 21, 2, 11, 3, 4, 7, 13 soluția este 2, 3, 7, 13 (2p)

Obs: Subiectele se rezolvă pe foaie, scris de mână.

Fiecare pagină, în colțul din dreapta sus, să conțină: nume prenume, grupa, numărul subiectului, numerotare pagină.

Subiectele se pot rezolva în orice ordine pe foaie.

Nu trebuie să copiați enunțul problemei (doar să indicați clar numărul problemei rezolvate)

Dacă nu se rezolvă subiectul eliminatoriu (problema 1) examenul scris este picat.

Înainte de expirarea timpului trebuie să trimiteți un singur fișier pdf, care conține poze de pe fiecare pagină de rezolvare. Pozele să fie cât mai clare, să aibă orientarea corectă în pdf, o poză per pagină de pdf.

Se corectează doar paginile trimise corect care se pot citi și au fost trimise până la timpul limită anunțat.

1. $\text{def } \text{chr} (l : \text{list}, x : \text{int}) :$

$st = 0$

$dr = \text{len}(l)$

$\text{while } dr - st > 1 :$

$m = (dr + st) // 2$

$\text{if } (l[m] > x)$

$dr = m$

$\text{if } (l[m] \leq x)$

$st = m$

$\text{return } m$

$\text{def } \text{chr} (l : \text{list}, x : \text{int}, st : \text{int}, dr : \text{int}) :$

$st = 0$

$dr = \text{len}(l)$

$\text{if } dr - st \leq 1 :$

$\text{return } st$

$m = (dr + st) // 2$

$\text{if } l[m] > x :$

$\text{return } \text{chr}(l, x, st, m)$

```

if l[m] ≤ x:
    return chr(l, x, m, dr)

```

2. """ Funcția verifică dacă n conține cifre pare
 : param n: numărul pe care-l verificăm
 : type n: int
 : return: True/False în funcție dacă există sau nu cifre pare
 : type return: bool
 : raises: ValueError dacă n e negativ sau 0
 """

```

def test_f():
    assert f(2) == True
    assert f(3) == False
    assert f(123) == True
    assert f(20) == True
    assert f(10) == True
    try:
        f(0)
    except:
        assert False

```

except ValueError:

assert True

test-f(1)

3. Time Complexity

$$T(S(l, \text{poz}=0)) = \begin{cases} 1, & \text{poz} = \text{len}(l) \\ 2T(S(l, \text{poz}+1)), & \text{otherwise} \end{cases}$$

$$T(S(l, 0)) = 2^1 T(S(l, 1))$$

$$2T(S(l, 1)) = 2^2 T(S(l, 2))$$

$$2^2 T(S(l, 2)) = 2^3 T(S(l, 3))$$

...

$$2^{K-1} T(S(l, \text{len}(l)-1)) = 2^K T(S(l, \text{len}(l))) = 2^K, \quad K = \text{len}(l) \in \Theta(2^n)$$

Space Complexity

Este $\Theta(1)$ deoarece nu se creează liste sau alte obiecte.

4. def f(lista: list) -> bool:

if len(lista) == 1:

return lista[0] % 2 == 0

mid = len(lista)

a = f (lista[:mij])

b = f (lista[mij:])

return a or b

