Fundamentele programării

1 Implementați căutare binara recursiv si nerecursiv. Subiect eliminatoriu. (1p)

2 Specificați si 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 si spațiu a următorului algoritm. (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 in 2 parți egale la fiecare pas. Ex: [1,3,4,5] -> Adevărat (2p).

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

Obs: Subiectele se rezolva pe foaie, scris de mana.

Fiecare pagina, in coltul din dreapta sus, sa conțină: nume prenume, grupa, numărul subiectului, numerotare pagina.

Subiectele se pot rezolva in orice ordine pe foaie.

Nu trebuie sa copiat enunțul problemei (doar sa indicați clar numărul problemei rezolvate)

Daca nu se rezolva subiectul eliminatoriu (problema 1) examenul scris este picat.

Înainte de expirarea timpului trebuie sa trimiteți un singur fișier pdf, care conține poze de pe fiecare pagina de rezolvare. Pozele sa fie cat mai clare, sa aibă orientarea corecta in pdf, o poza per pagina de pdf.

Se corectează doar paginile trimise corect care se pot citi si au fost trimise pana la timpul limita anunțat.

1.
```
def cbin(l: list, x: int):
    st = 0
    dr = len(l)
    while dr - st > 1:
        m = (dr + st) // 2
        if (l[mij] > x)
            dr = mij
        if (l[mij] ≤ x)
            st = mij
    return mij
```

```
def cbin(l: list, x: int, st: int, dr: int):
    st = 0
    dr = len(l)
    if dr - st <= 1:
        return st
    m = (dr + st) // 2
    if l[m] > x:
        return cbin(l, x, st, m)
```

```
    if l[m] <= x:
        return cbs(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)
        assert False
```

except ValueError:
    assert True

test_f()

3. Time Complexity

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

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

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

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

...

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

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
    mij = len(lista)

a = f(lista[:mij])
b = f(lista[mij:])
return a or b

5.

i	0	1	2	3	4	5	6
l	21	2	11	3	4	7	13
lgs	1	4	2	3	0	2	1
v	0	3	6	5	0	6	0

start = 1