Tutoriat 8 – Divide et Impera. Backtracking
- rezolvări -

1. Se citește o listă `l1` de numere sortată crescător. Citindu-se o listă nouă `l2`, afișați pentru fiecare număr din `l2` dacă se află în `l1`.

```
def bs( x, v ):
    if len(v) == 0:
        return False
    if len(v) == 1:
        if v[0] == x:
            return True
        return False
    mij = len(v) // 2
    if x == v[mij]:
        return True
    if x < v[mij]:
        return bs( x, v[:mij] )
    if x > v[mij]:
        return bs( x, v[mij + 1:] )

l1 = [ int(x) for x in input().split() ]
l2 = [ int(x) for x in input().split() ]

for elem in l2:
    print( f'{elem}: {bs( elem, l1 )}' )
```

2. Folosind metoda Divide et Impera, aflați elementul cu valoare maximă dintr-o listă citită de la tastatură.

```
def maxim( v ):
    if len(v) == 0:
        return 0
    if len(v) == 1:
        return v[0]
    mij = len(v) // 2
    return max( maxim( v[:mij] ), maxim( v[mij:] ) )

v = [ int(x) for x in input().split() ]
print( maxim(v) )
```

3. Scrieți subprogramul `mergeSort`, care, primind ca parametru o listă și o funcție cheie, sortează lista prin interclasare după cheia dată.

```
def mergeSort( v, f = lambda x: x ):
    if len(v) <= 1:
        return
    mij = len(v) // 2
    v1 = v[:mij]
    v2 = v[mij:]
    mergeSort( v1, f )
    mergeSort( v2, f )
    v[:] = merge( v1, v2, f )

def merge( v1, v2, f ):
    i = 0
    j = 0
    aux = []
    while i < len(v1) and j < len(v2):
        if f( v1[i] ) <= f( v2[j] ):
            aux.append( v1[i] )
            i += 1
        else:
            aux.append( v2[j] )
            j += 1
    while i < len(v1):
        aux.append( v1[i] )
        i += 1
    while j < len(v2):
        aux.append( v2[j] )
        j += 1
    return aux

v = [ int(x) for x in input().split() ]
nrDiv = lambda x: len( [ i for i in range(1, x + 1) if x % i == 0 ] )
mergeSort( v, nrDiv )
print(*v)
```

4. Se citește o listă de numere naturale, reprezentând valorile unor bancnote. Citindu-se o sumă `s`, generați toate modurile în care se poate achita suma dată, cu bancnotele puse la dispoziție. Se consideră că există un număr infinit de bancnote de fiecare tip.

Exemplu:

Intrare	Iesire
1 2 5 s = 5	1 1 1 1 1 1 1 1 2 1 2 2 5

```
def bkt( s, bancnote, sol = [] ):
    sol = sol[:]
    if sum(sol) == s:
        print( *sol )
        return
    if sum(sol) > s:
        return
    for b in bancnote:
        sol.append(b)
        if verif( sol ):
            bkt( s, bancnote, sol )
        sol = sol[:-1]

def verif( v ):
    ok = True
    for i in range( 1, len(v) ):
        if v[i] < v[i - 1]:
            ok = False
            break
    return ok

bancnote = [ int(x) for x in input('bancnote = ').split() ]
s = int(input('s = '))
bkt( s, bancnote )
```

5. Generați toate aranjamentele de `n` luate câte `k` ale unei mulțimi de `n` elemente, citită de la tastatură. Atât elementele listei, cât și `n` și `k` vor fi numere naturale, $0 \leq k \leq n$.

Exemplu:

Intrare	Iesire
7 1 25 k = 2	7 1 7 25 1 7 1 25 25 7 25 1

```
def bkt( l, k, sol = [] ):
    sol = sol[:]
    if len(sol) == k:
        print( *sol )
        return
    for elem in l:
        sol.append(elem)
        if verif( sol ):
            bkt( l, k, sol )
        sol = sol[:-1]
```

```
def verif( v ):
    return len( v ) == len( set( v ) )

l = [ int(x) for x in input('l = ').split() ]
k = int(input('k = '))
bkt( l, k )
```