

## SUBIECT 14

3. complexitatea TEMP:

def f(m):

l = []

for i in range (m\*m):

j = 1

while j < m:

l.append(j)

j = j \* 2

return l

while se execută de  $\log_2 n$  ori pt. că j parcurge poz. puteri de 2  
for se executa de  $n \cdot n$  ori

$$\Rightarrow T(n) = n \cdot n \cdot \log_2 n \in \Theta(n^2 \log n)$$

complexitatea spațiu:

este necesar spațiu suplimentar față de input deoarece pentru lista creată, care va avea  $n \cdot n \cdot \log_2 n$  poziții  $\Rightarrow \in \Theta(n^2 \log n)$

2. specificații:

Functia calculează suma cifrelor unui nr. pozitiv.

: param m: numărul pt. care se va calcula suma cifrelor

: type m: int

: return: suma cifrelor lui m

: type return: int

: raises: ValueError dacă m < 0

teste:

```
def test_f():
```

```
    m = -27
```

```
    try:
```

```
        f(m)
```

```
        assert False
```

```
    except ValueError:
```

```
        assert True
```

```
m = 1
```

```
assert f(1) == 1
```

```
m = 152
```

```
assert f(m) == 8
```

test\_f()

2.

$g(x) \rightarrow x = [1]$

$\text{print}(x[-1]) \rightarrow \text{print} 1$

$f(a) \rightarrow a = 2$

$\text{print} a \rightarrow \text{print} 1$  variabilele nu sunt mutabile!

$g(x) \rightarrow x = []$

$g(x) \rightarrow \text{ValueError} \rightarrow \text{print} E$

$\Rightarrow 19E$

4. mulțam facut - băda

## SUBIECTUL 13

3. def  $g(x)$ : $l = []$ 

for el in x:

 $k_i = 1$ while  $k_i < \text{len}(x)$  and  $el < 0$ : $l.append(k_i)$  $k_i = k_i * 2$ best case: toate el. dim. x sunt  $\geq 0$  $\Rightarrow$  for-ul se execută de m ori (lungimea listei)

while nu se execută

 $\Rightarrow T(m) = m \in \Theta(m)$ worst case: toate el. dim. x sunt  $< 0$  $\Rightarrow$  for-ul se execută de m oriwhile se execută de  $\log_2 m$  ori ( $k_i$  parcurge poz. puteri de 2) $\Rightarrow T(m) = m \cdot \log_2 m \in \Theta(m \log m)$ 

average case:

$k_i = 2^3$

$k_i = 2^1$

$k_i = 2^2$

 $\vdots$ 

$k_i = 2^m = m$

 $k_i = 2^{m+1} > m \Rightarrow$  se execută de m ori  $= \log_2 k_i$  $\Rightarrow T(m) =$ poate intra o dată, două ori, ...,  $\log_2 m$  ori

$\Rightarrow k_i = \log_2 m \Rightarrow m = 2^{k_i}$

??

2. Funcția verifică dacă o listă conține sau nu numere impare.

: param: l: lista în care se caută

: type l: list

: return: True dacă nu există nr. imparu și False altfel

: return type: bool

: raises: ValueError dacă lista este goală sau nu a fost inițializată

teste: cu lista goală, lista cu nr. impar, lista fără nr. impar

## 1. INSERTION SORT

for i in range (1, len (lista)):

    index = i - 1

    a = lista [i]

    while index >= 0 and a < lista [index]:

        lista [index + 1] = lista [index]

        index = index - 1

    lista [index + 1] = a

return lista

## 1. SELECTION SORT

```

def selection_sort(lista):
    for i in range(0, len(lista) - 1):
        index = i
        for j in range(i + 1, len(lista)):
            if lista[j] < lista[index]:
                index = j
        lista[index], lista[i] = lista[i], lista[index]
    return lista

```

## 2. Verifică dacă o listă conține sau nu numere pare.

- : param l:
- : type l:
- : return: nr. de numere pare din listă
- : type return: int
- : raises: ValueError dacă lista e goală sau nu a fost initializată

teste: cu lista goală, listă cu nr. paru, listă fără nr. pare

## 3. def g(l):

```

if len(l) > 1:
    aux = l[:len(l) // 2]
    g(aux)
    for el in l:
        print(el)

```

g(aux)

else:

print(len(l))

$$T(m) = \begin{cases} 1 & \text{daca } m \leq 1 \\ 2 \cdot T\left(\frac{m}{2}\right) + m & \text{altfel} \end{cases}$$

$$\begin{aligned} T(m) &= 2 \cdot T\left(\frac{m}{2}\right) + m = 2 \left(2 \cdot T\left(\frac{m}{4}\right) + \frac{m}{2}\right) + m = 2^2 T\left(\frac{m}{2^2}\right) + 2 \cdot m = \\ &= 2^2 \left(2 \cdot T\left(\frac{m}{8}\right) + \frac{m}{4}\right) + 2m = 2^3 T\left(\frac{m}{2^3}\right) + 3m \end{aligned}$$

$$T\left(\frac{m}{2}\right) = 2 \cdot T\left(\frac{m}{4}\right) + \frac{m}{2}$$

$$T\left(\frac{m}{4}\right) = 2 \cdot T\left(\frac{m}{8}\right) + \frac{m}{4}$$

:

merge pînă la  $\frac{m}{2^k}$  unde numitorul  $= m = 2^k \Rightarrow k = \log_2 m$

$$\Rightarrow T\left(\frac{m}{2^k}\right) = 1$$

$$\Rightarrow T(m) = 2^k \cdot T\left(\frac{m}{2^k}\right) + k \cdot m = 2^{\log_2 m} + \underbrace{\log_2 m \cdot m}_{\in \Theta(m)} \in \Theta(m + \log m) \subset \Theta(m)$$

$\frac{m}{2^k}$

complexitate spațiu:

rezultă o listă de  $\frac{m}{2^1}, \text{ apoi } \frac{m}{2^2}, \dots, 1$  elemente

$\Rightarrow$

3. def g(m)

$l = [ ]$

for i in range (m\*m):

$k_1 = 1$

while  $k_1 < m$ :

l.append (k)

$k_1 = k_1 + 2$

while merge de  $\log_2 m$  ori (creste cu puteri de 2)

for merge de  $m \cdot m$  ori

$$\Rightarrow T(m) = m + m \cdot \log_2 m \in \Theta(m^2 \log m)$$

complexitatea spatială:

se crează o listă de  $m \cdot m \cdot \log m$  elemente  $\Rightarrow \Theta(m^2 \log m)$

2. aux =  $e_1 + 1$

$$e_i - aux < 0 \Leftrightarrow e_i - e_1 - 1 < 0 \Rightarrow -1 < 0 \text{ "A"}$$

$$e_2 - e_1 < 0 \Leftrightarrow e_2 < e_1 \quad e_1 - e_0 < 0 \Rightarrow e_1 < e_0$$

$\Rightarrow$  Funcția verifică dacă un sir este strict descrescător

⋮

1. MERGE SORT

def merge\_sort (lista):

if len(lista) == 1:

return lista

mid = len(lista) // 2

st = merge\_sort (lista[:mid])

dr = merge\_sort (lista[mid:])

apoi interclasare st, dr

# SUBJECT 10

## 1. BINARY SEARCH

### 2. RECURSIV

```

def bs(lista, x):
    if len(lista) == 1:
        if lista[0] == x:
            return 0
    st = 0
    dr = len(lista) - 1
    while st <= dr:
        mid = (st + dr) // 2
        if x == lista[mid]:
            return mid
        if x < lista[mid]:
            dr = mid - 1
        else:
            st = mid + 1
    return -1

```

### 2. RECURSIV

```

def bs(lista, x, st=0, dr=-2):
    if dr == -2:
        dr = len(lista) - 1
    if st > dr:
        return -1
    mid = (st + dr) // 2
    if lista[mid] == x:
        return mid
    if x < lista[mid]:
        bs(lista, x, st, mid - 1)
    else:
        bs(lista, x, mid + 1, dr)

```

### 3. def g(l):

```

if len(l) > 1:
    m = len(l) // 2
    g(l[:m]) → T(½)
    for el in l[m:]:
        print(el)
    g(l[m:]) → T(½)

```

else :

print (len(l))

$$T(m) = \begin{cases} 1 & m = 1 \\ 2T(\frac{m}{2}) + m & \text{otherwise} \end{cases}$$

$$\Rightarrow T(m) = 2^k \cdot T\left(\frac{m}{2^k}\right) + km$$

$$\text{pp. } 2^k = m \Rightarrow k = \log_2 m$$

$$T(1) = 1 \Rightarrow T(m) = 2^{\log_2 m} + \log_2 m \cdot m$$

$$\in \Theta(m \cdot \log m)$$

$$2. e_1 - e_2 < 0 \Rightarrow -e_1 > e_2 \text{ "A"}$$

$$e_1 - e_2 < 0 \Rightarrow e_1 < e_2$$

Functia verifică dacă lista este strict crescătoare

### SUBIECT 9

1. Quicksort

3. def f(m):

$l = [ ]$

for i in range ( $m^2$ ):  $\rightarrow m^2$  ori

$j = 1$

while  $j < m$ :  $\rightarrow \log m$  ori

$l.append(j)$

$j = j * 2$

return l

$$\Rightarrow T(m) = m^2 \log m \text{ complexitate timp}$$

Spatiu: creaza o lista de  $m^2$  elem  $\Rightarrow \Theta(m^2 \log m)$

2. Calcularea sumă a cifrelor unui nr.

## SUBIECT 8

$$3. \quad T(m) = \begin{cases} 1 & m=0 \\ 2 \cdot T(m-1) & \text{altfel} \end{cases}$$

def  $s(l, pos=0)$   $\rightarrow T(m)$

if  $pos < len(l)$ :

$$a = s(l, pos+1) \rightarrow T(m-1)$$

$$b = s(l, pos+1) \rightarrow T(m-1)$$

return  $a+b$

else:

return 1

$$\Rightarrow T(m) = 2T(m-1) = 2 \cdot 2T(m-2) = 2^2 \cdot T(m-2)$$

$$T(m-1) = 2T(m-2)$$

:

$$T(m_0) = 2^{m_0} \cdot T(m-m_0) = 2^m$$

$\Rightarrow$  complexitatea timp  $T(m_0) \in \Theta(2^m)$

complexitatea spațiu:

crează mereu o nouă variabilă noi

2. Verifică dacă un număr are minimum de cifre pară.

## 1. BINARY SEARCH

### 1. NERECURSIV

```
def bs(lst, x):  
    st = 0  
    dr = len(lst) - 1  
    mid = (st + dr) // 2  
    while (st < dr):  
        mid = (st + dr) // 2  
        if lst[mid] == x:  
            return mid  
        if x < lst[mid]:  
            dr = mid - 1  
        else : st = mid + 1  
  
    return -1  
  
2. RECURSIV  
def bs(lista, x, st=0, dr=-2):  
    if dr == -2:  
        dr = len(lista) - 1  
    if st > dr:  
        return -1  
    mid = (st + dr) // 2.  
    if lista[mid] == x:  
        return mid  
    if x < lista[mid]:  
        bs(lista, x, st, mid-1)  
    else :  
        bs(lista, x, mid+1, dr)
```

# SUBJECT 7

## 1. INSERTION SORT

def IS(lst) :

for i in range(1, len(lst)):

    index = i - 1

    a = ~~lst[index]~~ lst[i]

    while index >= 0 and a < lst[index]:

        lst[index + 1] = lst[index]

        index = index - 1

    lst[index + 1] = a

return lst

## 3. def f(l):

if len(l) == 1       → 1

    return l[0]

if l[0] == 0:       → 1

    return 0

return l[0] \* f(l[1:])   → T(m-1)+2

cas favorabil: este o pe prima poz

⇒ T(m) ∈ Θ(1)

cas dezfavorabil: lista fără 0 -uri

$$T(m) = \begin{cases} 1 & m=0 \\ \end{cases}$$

$$\begin{cases} T(m-1)+2 & \text{altfel} \end{cases}$$

$$T(m) \rightarrow T(m-1)+2 \rightarrow T(m-2)+2 \cdot 2 = \dots = T(m-m) + m \cdot 2 = 1 + m \cdot 2$$

$$T(m-1) = T(m-2)+2$$

$$Θ(m)$$

2.  $\ell = [0, 1, \dots, m-1]$  invers

$$\ell[2] = \ell[1] + \ell[2] = (m-1) + (m-2)$$

Calcularea sumă nr. până la  $m-1$ .

SUBIECT 6

## PROGRAMARE DINAMICĂ:

\* longest increasing subsequence

def funcție (lista):

lungime = 1

final = 0

l = 1

p = [-1]

for i in range (1, len(lista)) :

l.append(1)

p.append(-1)

for j in range (i-1, -1, -1) :

if  $a[i] > a[j]$  and  $l[j+1] > l[i]$  :

$l[i] = l[j] + 1$

$p[i] = j$

if  $l[i] > \text{lungime}$ :

final = i

lungime = l[i]

rez = []

i = final

while  $i \neq -1$ :

rez.append(a[i])

i = p[i]

rez.reverse()

return rez