# Portofoliu Algoritmi și complexitate

Pavel Andrei

## Laborator 1

15. O colecție de $n$ monede sunt identice, cu excepția uneia, falsă, care are greutatea mai mică decât a celorlalte. Propuneți o strategie pentru a identifica moneda falsă folosind o balanță simplă și cât mai puține cântăriri.

```python
import random, math
# The values aren't important, what matters is that fake < real
realWeight = 40
fakeWeight = 35

length = int(input("Please enter length: "))

coins = [realWeight] * length
coins[random.randrange(length)] = fakeWeight

bigPile = coins
steps = 0

while len(bigPile) > 1:
    steps += 1
    print("Step %d:" % steps)
    #we split the big pile in 3 piles with the same size ±1
    pileSize = len(bigPile) // 3 + (len(bigPile) % 3 == 2)
    leftPile = bigPile[0:pileSize]
    rightPile = bigPile[pileSize:pileSize*2]
    leftoverPile = bigPile[pileSize*2:]
    print("left", leftPile)
    print("right", rightPile)
    print("leftover", leftoverPile)
    difference = 0
    for i in range(pileSize):
        difference += rightPile[i] - leftPile[i]

    if difference > 0: bigPile = leftPile
    elif difference < 0: bigPile = rightPile
    else: bigPile = leftoverPile
    print()

print("Took", steps, "steps to find the fake coin, expected",
      math.ceil(math.log(length, 3)))
```

```
$ python3 src.py
Please enter length: 15
Step 1:
left [40, 40, 40, 40, 40]
right [40, 40, 40, 40, 40]
leftover [40, 40, 35, 40, 40]

Step 2:
left [40, 40]
right [35, 40]
leftover [40]

Step 3:
left [35]
right [40]
leftover []

Took 3 steps to find the fake coin, expected 3
```

16. Bucătarul unui restaurant a pregătit clătite și le-a stivuit pe o farfurie. Fiind începător, clătitele nu au ieșit la fel, având diametre diferite, iar farfuria arată destul de rău. Bucătarul șef a vrut să-i dea o lecție și i-a dat sarcina de a rearanja (cea cu diametrul cel mai mare să fie prima pe farfurie, apoi cea cu diametrul imediat următor ca mărime ș.a.m.d.) folosind doar o spatulă. Ce strategie să adopte?

```python
print("Please insert a space separated list of pancake diameters:\n(",
      end="")
strs = input().split(' ')
# index 0 represents the topmost pancake
pancakes = [int(num) for num in reversed(strs) if num != ""]

def draw(msg, vec, spatulaIndex = -1):
    print(msg + "(", end="")
    i = len(vec)
    if spatulaIndex > 0:
        for i in range(i-1, spatulaIndex-1, -1): print(vec[i], end=" ")
        print("|", end="")

    for i in range(i-1, -1, -1): print(vec[i], end=" ")
    print()

flips = 0
def flip(vec, index):
    global flips
    flips += 1
    draw("flip %2d: " % (flips), pancakes, index)
    for i in range(0, index//2):
        t = vec[i]
        vec[i] = vec[index-i-1]
        vec[index-i-1] = t

```

```
for bottom in range(len(pancakes), 0, -1):
    spatulaIndex = 0
    for i in range(0, bottom):
        if pancakes[i] >= pancakes[spatulaIndex]: spatulaIndex = i
    spatulaIndex += 1
    # if the biggest is at the bottom we do nothing
    if spatulaIndex == bottom: continue
    # if the biggest is already at the top we don't have to flip it
    if spatulaIndex != 1: flip(pancakes, spatulaIndex)

    flip(pancakes, bottom)

draw("", pancakes)
print("Done in %d flips" % flips)
```

```
$ python3 src.py
Please insert a space separated list of pancake diameters:
(5 9 4 3 7 2 8 1
flip  1: (5 |9 4 3 7 2 8 1
flip  2: (|5 1 8 2 7 3 4 9
flip  3: (9 4 3 7 2 |8 1 5
flip  4: (9 |4 3 7 2 5 1 8
flip  5: (9 8 1 5 2 |7 3 4
flip  6: (9 8 |1 5 2 4 3 7
flip  7: (9 8 7 3 4 2 |5 1
flip  8: (9 8 7 |3 4 2 1 5
flip  9: (9 8 7 5 1 2 |4 3
flip 10: (9 8 7 5 |1 2 3 4
(9 8 7 5 4 3 2 1
Done in 10 flips

$ python3 src.py
Please insert a space separated list of pancake diameters:
(4 3 2 1
(4 3 2 1
Done in 0 flips

$ python3 src.py
Please insert a space separated list of pancake diameters:
(3 3 1 4 3
flip  1: (3 3 1 |4 3
flip  2: (|3 3 1 3 4
flip  3: (4 3 1 |3 3
flip  4: (4 3 |1 3 3
(4 3 3 3 1
Done in 4 flips
```

## Laborator 2

10. Demonstrați corectitudinea algoritmului de determinare a valorii obținute prin inversarea ordinii cifrelor unui număr natural.

```
n = int(input("Please insert n: "))

res = 0
i = 0

while n != 0:
    res *= 10
    res += n % 10
    n //= 10
    i += 1

print("Result:", res)
```

```
$ python3 reversed.py
Please insert n: 1234
Result: 4321

$ python3 reversed.py
Please insert n: 2400
Result: 42
```

I. Parțial corectitudinea

Considerăm serțiunile de intrare și ieșire:

$P_{in} = \left\{ n = \sum_{j=0}^{k} c_j 10^j;\ c_j \in \overline{0,9},\ \forall j \in \overline{0,k};\ c_k \neq 0 \right\},$

$P_{out} = \left\{ res = \sum_{j=0}^{k} c_{k-j} 10^j \right\}.$

Alegem proprietatea:

$I = \left\{ n = \sum_{j=0}^{k-i} c_{i+j} 10^j; res = \sum_{j=0}^{i-1} c_{i-1-j} 10^j \right\}.$

La intrarea in buclă:

$i = 0$

$n = \sum_{j=0}^{k} c_j 10^j$

Deci propoziția $I = \left\{ n = \sum_{j=0}^{k} c_j 10^j; res = \sum_{j=0}^{-1} c_{-1-j} 10^j = 0 \right\}$ este adevărată.

Arătăm că propoziția $I$ este invariantă.

Presupunem $I$ adevărata la începutul iterației și $n \neq 0$; demonstrăm $I$ adevărata la sfârșitul iterației.

$n = \sum_{j=0}^{n-i} c_{i+j} 10^j;\ res = \sum_{j=0}^{i-1} c_{i-1-j} 10^j$

```
7   res *= 10
```

$$res = \left(\sum_{j=0}^{i-1} c_{i-1-j}10^j\right) \cdot 10 = \sum_{j=0}^{i-1} c_{i-1-j}10^{j+1} = \sum_{j=1}^{i} c_{i-j}10^j$$

```
8   res += n % 10
```

$$res = \left(\sum_{j=1}^{i} c_{i-j}10^j\right) + c_i = \left(\sum_{j=1}^{i} c_{i-j}10^j\right) + c_{i-0}10^0 = \sum_{j=0}^{i} c_{i-j}10^j$$

```
9   n //= 10
```

$$n = \left[\left(\sum_{j=0}^{k-i} c_{i+j}10^j\right) / \ 10\right] = \left[\sum_{j=0}^{k-i} c_{i+j}10^{j-1}\right] = \left[\sum_{j=1}^{k-i} c_{i+j}10^{j-1}\right] + \left[c_i 10^{-1}\right]$$

Cum $0 \leq c_i \leq 9 \implies 0 \leq c_i 10^{-1} \leq 0.9 \implies \left[c_i 10^{-1}\right] = 0.$

Deci $n = \left[\sum_{j=1}^{k-i} c_{i+j}10^{j-1}\right] = \sum_{j=1}^{k-i} c_{i+j}10^{j-1} = \sum_{j=0}^{k-i-1} c_{i+j+1}10^j.$

```
10  i += 1
```

Scriem *res* și $n$ în funcție de noul $i$. Deci $i$ devine $i-1$.

$res = \sum_{j=0}^{i-1} c_{i-1-j}10^j$

$n = \sum_{j=0}^{k-(i-1)-1} c_{i-1+j+1}10^j = \sum_{j=0}^{k-i} c_{i+j}10^j$

Deci $I$ adevărata și la sfârșitul iterației.

La ieșirea din buclă:

$i = k+1$

$n = \sum_{j=0}^{k-(k+1)} c_{k+1+j}10^j = \sum_{j=0}^{-1} c_{k+1+j}10^j = 0$

$res = \sum_{j=0}^{k+1-1} c_{k+1-1-j}10^j = \sum_{j=0}^{k} c_{k-j}10^j$

Deci $P_{out} = \left\{res = \sum_{j=0}^{k} c_{k-j}10^j\right\}$ adevărată.

În concluzie algoritmului este parțial corect.

II. Total corectitudinea

Considerăm funcția $t : \mathbb{N} \to \mathbb{N}$, $t(i) = k+1-i$ $t(i+1)-t(i) = k+1-(i+1)-(k+1-i) = -1 < 0$, deci $t$ monoton strict descrescătoare.

$t(i) = 0 \iff i = k+1 \iff n = \sum_{j=0}^{-1} c_{k+1+j}10^j = 0 \iff$ condiția de ieșire din buclă.

În concluzie algoritmului este total corect.

## Laborator 3

10. Considerăm o secvență $x = (x_0,...,x_{n-1})$ de $n$ numere întregi, cu măcar un element pozitiv. O subsecvență a șirului este de forma $(x_i, x_{i+1}, ..., x_j)$, cu $0 \leq i \leq j \leq n-1$, iar suma subsecvenței este suma elementelor componentelor sale. Descrieți un algoritm pentru a determina subsecvența de sumă maximă. Estimați timpul de execuție al algoritmului, precizând operația dominantă.

```
1  print("Please insert the sequence: ", end="")
2  strs = input().split(' ')
3  v = [int(num) for num in strs if num != ""]
4  n = len(v)
5  # python way of defining a n-dimensional list initialized to 0
6  sub_sums = [0 for i in range(0, n)]
7
8  best = (0, 0)
9  best_sum = 0
10 for i in range(0, n):
11     sub_sums[i] = v[i]
12     best_end_index = i
13     # after this loop v[j] = (sum from k=i to j of v[k])
14     for j in range(i+1, n):
15         sub_sums[j] = sub_sums[j-1] + v[j]
16         if sub_sums[j] > sub_sums[best_end_index]:
17             best_end_index = j
18     if sub_sums[best_end_index] > best_sum:
19         best_sum = sub_sums[best_end_index]
20         best = (i, best_end_index)
21
22 print("Best with a sum of", best_sum, "is: (x%d,...,x%d)" % best)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 4
Best with a sum of 10 is: (x0,...,x3)

$ python3 src.py
Please insert the sequence: 1 -2 3 4
Best with a sum of 7 is: (x2,...,x3)

$ python3 src.py
Please insert the sequence: 1 2 -3 4
Best with a sum of 4 is: (x0,...,x3)
```

Considerăm operația de baza ca fiind compararea elementelor tabloului `v` (liniile 16 și 18).
Notăm $T_l(n) :=$ timpul total de execuție al liniei $l$; $T(n) :=$ timpul de execuție total.

$$T_{16}(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1} ((n-1) - i) = n(n-1) - \sum_{i=0}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} = \frac{n(n-1)}{2}$$

$$T_{18}(n) = \sum_{i=0}^{n-1} 1 = n$$

$$T(n) = \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2}$$

## Laborator 4

8. Considerăm o secvența $x = (x_0, ..., x_{n-1})$ de $n$ numere întregi. Generați tabloul $f = (f_0, ..., f_{n-1})$, cu $f_i = \sum_{j=0}^{i} x_j$, printr-un algoritm de complexitate liniară.

```
print("Please insert the sequence: ", end="")
strs = input().split(' ')
x = [int(num) for num in strs if num != ""]
n = len(x)
f = [0 for i in range(n)]

f[0] = x[0]
for i in range(1, n):
    f[i] = f[i-1] + x[i]

print(f)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 0 -1 5
[1, 3, 6, 6, 5, 10]
```

9. Considerăm un tablou de valori întregi $x = (x_0, ..., x_{n-1})$ și o valoare dată, s. Să se verifice daca există cel puțin doi indici $i$ și $j$ (nu neapărat distincți) cu proprietatea că $x_i = x_j = s$. Analizați complexitatea algoritmului propus.

```
print("Please insert the sequence: ", end="")
strs = input().split(' ')
x = [int(num) for num in strs if str != ""]
print("Please insert s: ", end="")
s = int(input())

def f(x):
    for i in range(0, len(x)):
        for j in range(i, len(x)):
            if x[i] + x[j] == s:
                print ("Found %d + %d = %d " % (x[i] , x[j], s))
                return True
    print("Not found")
    return False
f(x)
```

```
$ python3 src.py
Please insert the sequence: 1 2 3 0 -1 5
Please insert s: 9
Not found

$ python3 src.py
Please insert the sequence: 1 2 3 0 5 -1
Please insert s: 7
Found 2 + 5 = 7
```

## Laborator 5

4. (*Shaker sort*) modificând algoritmul de sortare prin interschimbarea elementelor vecine, sortați elementele unui tablou, astfel încât, la fiecare pas, să se plaseze pe pozițiile finale câte două elemente: minimul, respectiv maximul din subtabloul parcurs la pasul respectiv.

```
print("Please insert the array: ", end="")
strs = input().split(' ')
v = [int(num) for num in strs if num != ""]

def impl(start, end, step):
    sorted = True
    for i in range(start, end, step):
        if v[i] > v[i+1]:
            t = v[i]
            v[i] = v[i+1]
            v[i+1] = t
            sorted = False
    return sorted

begin = 0
end = len(v) - 1

while True:
    if impl(begin, end, 1): break
    if impl(end-1, begin-1, -1): break

    end -= 1
    begin += 1

print(v)
```

```
$ python3 shaker_sort.py
Please insert the array: 6 5 3 1 8 7 2 4 0 9
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

$ python3 shaker_sort.py
Please insert the array: 4 1 0 2 7 3 9 8 5 6
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

$ python3 shaker_sort.py
Please insert the array: 9 4 3 0 5 6 1 8 7 0
[0, 0, 1, 3, 4, 5, 6, 7, 8, 9]

$ python3 shaker_sort.py
Please insert the array: 40 3 43 95 9 2 4 0
[0, 2, 3, 4, 9, 40, 43, 95]
```

5. (*Counting sort* - sortare prin numărare) Considerăm un tablou $x$ de dimensiune $n$, cu elemente din mulțimea $\{0, 1, 2,...,m\}$. Pentru sortarea unui astfel de tablou poate fi descris un algoritm de sortare de complexitate liniară, dacă $m$ nu este semnificativ mai mare ca $n$. Pașii algoritmul sunt:

(a) se construiește tabloul $f[0..m]$ al frecvențelor de apariție a elementelor tabloului $x$ ($f_i$ reprezintă de câte ori apare valoarea $i$ în tabloul $x$, $i = 0,...,m$);

(b) se calculează tabloul frecvențelor cumulate $fc[0..m]$, $fc_i = \sum_{j=0}^{i} f_j$, $i = 0,...,m$;

(c) se folosește tabloul frecvențelor cumulate pentru a construi tabloul ordonat.

Descrieți algoritmul de sortare prin numărare. Care este complexitatea acestuia?

```python
print("Please insert the array: ", end="")
x = [int(num) for num in (input().split()) if num != ""]
n = len(x)
m = max(x) + 1

f = [0 for i in range(m)]
output = [0 for i in range(n)]

for i in x: f[i] += 1
print("f:", f)

for i in range(1, m): f[i] = f[i-1] + f[i]

print("fc:", f)
for i in range(n):
    val = x[i]
    f[val] -= 1
    output[f[val]] = val

print("output:", output)
```

```
$ python3 counting_sort.py
Please insert the array: 0 1 2 2 1 0 2 1 2 4
f: [2, 3, 4, 0, 1]
fc: [2, 5, 9, 9, 10]
output: [0, 0, 1, 1, 1, 2, 2, 2, 2, 4]

$ python3 counting_sort.py
Please insert the array: 1 2 2 1 2 1 2 4
f: [0, 3, 4, 0, 1]
fc: [0, 3, 7, 7, 8]
output: [1, 1, 1, 2, 2, 2, 2, 4]
```

Considerăm atribuirile în vectori ca fiind operațiile de bază (ignorăm inițializările).
Notăm $T_l :=$ timpul total de execuție al liniei $l$; $T(n, m) :=$ timpul de execuție total.
$T_9 = n$; $T_{12} = m - 1$; $T_{17} = n$; $T_{18} = n$;
$T(n, m) = 3n + m - 1$.

$T \in O(n + m)$.

6. (*Radix sort* - sortare pe baza cifrelor) Considerăm un tablou $x$ de dimensiune $n$, cu elemente numere naturale de cel mult k cifre. Algoritmul de sortare este bazat pe următoarea idee: folosind counting sort, se ordonează tabloul în raport cu cifra cea mai puțin semnificativă a fiecărui număr, apoi se sortează în raport cu cifra de rang imediat superior ș.a.m.d., până de ajunge la cifra cea mai semnificativă.
Descrieți algoritmul radix sort. Care este complexitatea acestuia?

```
1  print("Please insert the array: ", end="")
2  x = [int(num) for num in (input().split()) if num != ""]
3  n = len(x)
4  max_x = max(x)
5
6  f = [0 for i in range(10)]
7  output = [0 for i in range(n)]
8
9  pow10 = 1
10 while max_x > 0:
11     def getDigit(num): return (num // pow10) % 10
12
13     for i in range(10): f[i] = 0
14     for i in x: f[getDigit(i)] += 1
15
16     for i in range(1, 10): f[i] += f[i-1]
17
18     for i in range(n - 1, -1, -1):
19         index = getDigit(x[i])
20         f[index] -= 1
21         output[f[index]] = x[i]
22
23     #output becomes new input
24     for i in range(n): x[i] = output[i]
25
26     pow10 *= 10
27     max_x //= 10
28
29 print("output:", output)
```

```
$ python3 radix_sort.py
Please insert the array: 3 2 4 23 427 459 56 90
output: [2, 3, 4, 23, 56, 90, 427, 459]

$ python3 radix_sort.py
Please insert the array: 89568 23 123 2 1 4 45 499
output: [1, 2, 4, 23, 45, 123, 499, 89568]
```

Considerăm atribuirile în vectori ca fiind operațiile de bază (ignorăm inițializările).

Notăm $k = log10(max(x)) + 1$; $T_l :=$ timpul total de execuție al liniei $l$; $T(n,k) :=$ timpul de execuție total.

$T_{13} = 10k$; $T_{14} = kn$; $T_{16} = 9k$; $T_{20} = kn$; $T_{21} = kn$;

$T(n,k) = 3kn + 19k$.

$T \in O(kn)$.