

## **Curs 11: Sortări**

- **Algoritmi de sortare**
  - metoda bulelor, quick-sort, merge sort
- **Sortare in Python: sort, sorted**
  - parametrii: poziționali, prin nume, implicita, variabil
  - list comprehension, funcții lambda

## **Curs 10-11: Căutări - sortări**

- **Căutări**
- **Algoritmi de sortare: selecție, selecție directă, inserție**

## Metoda bulelor - Bubble sort

- ‡ Compară elemente consecutive, dacă nu sunt în ordinea dorită le interschimbă.
- ‡ Procesul de comparare continuă până când nu mai avem elemente consecutive ce trebuie interschimbate (toate perechile respectă relația de ordine dată).

## Sortare prin metoda bulelor

```
def bubbleSort(l):  
    sorted = False  
    while not sorted:  
        sorted = True # assume the list is already sorted  
        for i in range(len(l)-1):  
            if l[i+1]<l[i]:  
                l[i], l[i + 1] = l[i + 1], l[i]  
                sorted = False # the list is not sorted yet
```

```
def bubbleSort2(l):  
    for j in range(1,len(l)):  
        for i in range(len(l)-j):  
            if l[i+1]<l[i]:  
                l[i], l[i + 1] = l[i + 1], l[i]
```

## **Complexitate metoda bulelor**

**Caz favorabil:**  $\theta(n)$  . Lista este sortată

**Caz defavorabil:**  $\theta(n^2)$  . Lista este sortată descrescător

**Caz mediu**  $\theta(n^2)$  .

**Complexitate generală** este  $O(n^2)$

**Complexitate ca spațiu adițional de memorie** este  $\theta(1)$  .

♣ este un algoritm de sortare *in-place* .

# QuickSort

Introdus de Sir Charles Antony Richard (Tony) **Hoare** in 1959

Bazat pe “divide and conquer”

- ‡ **Divide**: se împarte lista în 2 astfel încât elementele din dreapta pivotului sunt mai mici decât elementele din stânga pivotului.
- ‡ **Conquer**: se sortează cele două subliste
- ‡ **Combine**: trivial – dacă partiționarea se face în același listă

Partiționare: re-aranjarea elementelor astfel încât elementul numit *pivot* ocupă locul final în secvență. Dacă poziția pivotului este  $i$  :

$$k_j \leq k_i \leq k_l, \text{ for } Left \leq j < i < l \leq Right$$

# Quick-Sort

```
def partition(l, left, right):  
    """  
    Split the values:  
        smaller pivot greater  
    return pivot position  
    post: left we have < pivot  
        right we have > pivot  
    """  
    pivot = l[left]  
    i = left  
    j = right  
    while i!=j:  
        while l[j]>=pivot and i<j:  
            j = j-1  
        l[i] = l[j]  
        while l[i]<=pivot and i<j:  
            i = i+1  
        l[j] = l[i]  
    l[i] = pivot  
    return i
```

```
def quickSortRec(l, left, right):  
  
    #partition the list  
    pos = partition(l, left, right)  
  
    #order the left part  
    if left<pos-1:  
        quickSortRec(l, left, pos-1)  
  
    #order the right part  
    if pos+1<right:  
        quickSortRec(l, pos+1, right)
```

## QuickSort – complexitate timp de execuție

Timpul de execuție depinde de distribuția partiționării (câte elemente sunt mai mici decât pivotul câte sunt mai mari)

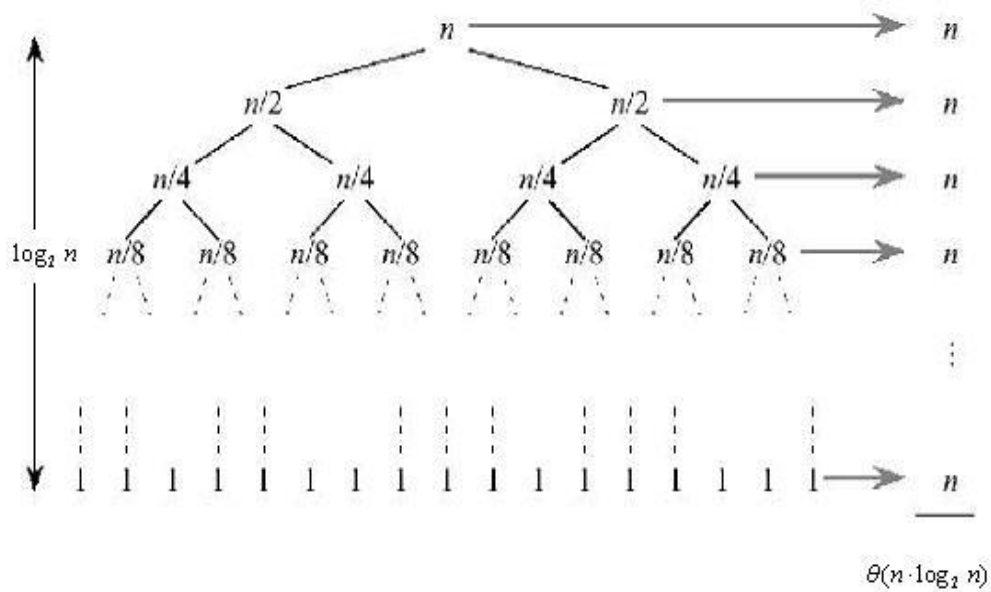
Partiționarea necesită timp linear.

**Caz favorabil:**, partiționarea exact la mijloc (numere mai mici ca pivotul = cu numere mai mari ca pivotul):

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \theta(n)$$

Complexitatea este  $\Theta(n \cdot \log n)$ .

# QuickSort – Caz favorabil

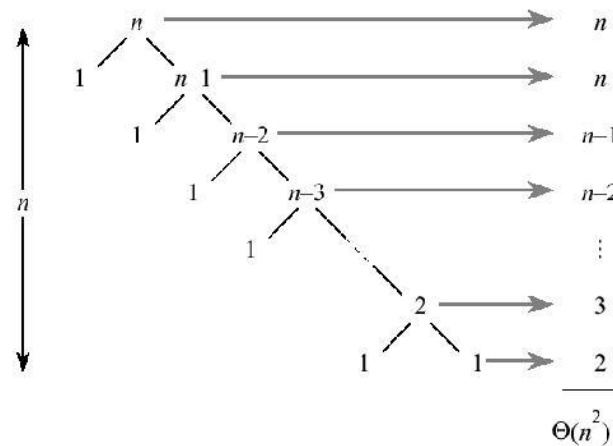




# QuickSort – Caz defavorabil

Partiționarea tot timpul rezultă într-o partiție cu un singur element și o partiție cu  $n-1$  elemente

$$T(n) = T(1) + T(n-1) + \theta(n) = T(n-1) + \theta(n) = \sum_{k=1}^n \theta(k) \in \theta(n^2)$$



caz defavorabil: dacă elementele sunt în ordine inversă

# QuickSort – Caz mediu

Se alternează cazurile:

♣ caz favorabil (lucky) cu complexitatea  $\Theta(n \cdot \log n)$  (notăm cu  $L$ )

♣ caz defavorabil (unlucky) cu complexitatea  $\theta(n^2)$  (notăm cu  $U$ ).

Avem recurența:

$$\begin{cases} L(n) = 2 \cdot U\left(\frac{n}{2}\right) + \theta(n) & \text{lucky case} \\ U(n) = L(n-1) + \theta(n) & \text{unlucky case} \end{cases}$$

Rezultă

$$L(n) = 2 \cdot \left( L\left(\frac{n}{2} - 1\right) + \theta\left(\frac{n}{2}\right) \right) + \theta(n) = 2 \cdot L\left(\frac{n}{2} - 1\right) + \theta(n) = \theta(n \cdot \log_2 n),$$

Complexitatea caz mediu:  $T(n) = L(n) \in \theta(n \cdot \log_2 n)$ .

## Coplexitatea ca timp de execuție pentru sortări:

### Complexity

Algorithm	worst-case	average
SelectionSort	$\theta(n^2)$	$\theta(n^2)$
InsertionSort	$\theta(n^2)$	$\theta(n^2)$
BubbleSort	$\theta(n^2)$	$\theta(n^2)$
QuickSort	$\theta(n^2)$	$\mathcal{O}(n \cdot \log n)$

# Python - Quick-Sort

```
def quickSort(list):  
    """  
    Quicksort using list comprehensions  
    return a new list  
    """  
    if len(list) <= 1:  
        return list  
    pivot = list.pop()  
    lesser = quickSort([x for x in list if x < pivot])  
    greater = quickSort([x for x in list if x >= pivot])  
    return lesser + [pivot] + greater
```

# List comprehensions – generatoare de liste

```
[x for x in list if x < pivot]
rez = []
for x in list:
    if x < pivot:
        rez.append(x)
```

- Variantă concisă de a crea liste
- creează liste unde elementele listei rezultă din operații asupra unor elemente dintr-o altă secvență
- paranteze drepte conținând o expresie urmată de o clauză **for** , apoi zero sau mai multe clauze **for** sau **if**

## *Python – Parametrii opționali parametrii cu nume*

- Putem avea parametrii cu valori default;

```
def f(a=7, b = [], c="adsdsa") :
```

- Dacă se apelează metoda fără parametru actual se vor folosi valorile default

```
def f(a=7, b = [], c="adsdsa") :  
    print (a)  
    print (b)  
    print (c)
```

**Console:**

```
7  
[]  
adsdsa
```

f ()

- Argumentele se pot specifica în orice ordine

```
f(b=[1, 2], c="abc", a=20)
```

**Console:**

```
20  
[1, 2]  
abc
```

- Parametrii formali se adaugă într-un dicționar (namespace)
- Trebuie oferit o valoare actuală pentru fiecare parametru formal - prin orice metodă: standard, prin nume, default value

## *Tipuri de parametri – cum specificăm parametru actual*

**positional-or-keyword** : parametru poate fi transmis prin poziție sau prin nume (keyword)

```
def func(foo, bar=None):
```

**keyword-only** : parametru poate fi transmis doar specificând numele

```
def func(arg, *, kw_only1, kw_only2):
```

Tot ce apare după \* se poate transmite doar prin nume

**var-positional** : se pot transmite un număr arbitrar de parametri poziționali

```
def func(*args):
```

Valorile transmise se pot accesa folosind args, args este un tuplu

**var-keyword**: un număr arbitrar de parametri prin nume

```
def func(**args):
```

Valorile transmise se pot accesa folosind args, args este un dicționar

## ***Sortare în python - list.sort() / funcție build in : sorted***

`sort(*, key=None, reverse=None)`

Sortează folosind operatorul <. Lista curentă este sortată (nu se creează o altă listă)

key – o funcție cu un argument care calculează o valoare pentru fiecare element, ordonarea se face după valoarea cheii. În loc de  $o1 < o2$  se face  $key(o1) < key(o2)$

reverse – true daca vrem sa sortăm descrescător

```
l.sort()
print (l)
```

```
l.sort(reverse=True)
print (l)
```

`sorted(iterable[, key][, reverse])`

Returnează lista sortată

```
l = sorted([1, 7, 3, 2, 5, 4])
print (l)
```

```
def keyF(o1):
    return o1.name
```

```
l = sorted([1, 7, 3, 2, 5, 4], reverse=True)
print (l)
ls = sorted(l, keyF)
```

## ***Sort stability***

***Stabil (stable)*** – dacă avem mai multe elemente cu același cheie, se menține ordinea inițială



## Python – funcții lambda (anonymous functions, lambda form)

- ⤴ Folosind `lambda` putem crea mici funcții

```
lambda x:x+7
```

- ⤴ Funcțiile lambda pot fi folosite oriunde e nevoie de un obiect funcție

```
def f(x):
    return x+7
print ( f(5) )
print ( (lambda x:x+7)(5) )
```

- ⤴ Putem avea doar o expresie.
- ⤴ Sunt o metodă convenientă de a crea funcții mici.
- ⤴ Similar cu funcțiile definite în interiorul altor funcții, funcțiile lambda pot referi variabile din namespace

```
l = []
l.append(MyClass(2, "a"))
l.append(MyClass(7, "d"))
l.append(MyClass(1, "c"))
l.append(MyClass(6, "b"))
#sort on name
ls = sorted(l, key=lambda o:o.name)
for x in ls:
    print (x)
```

```
l = []
l.append(MyClass(2, "a"))
l.append(MyClass(7, "d"))
l.append(MyClass(1, "c"))
l.append(MyClass(6, "b"))
#sort on id
ls = sorted(l, key=lambda o:o.id)
for x in ls:
    print (x)
```

## *MergeSort*

Introdus de John von Neumann în 1945

Bazat pe “divide and conquer”.

Secvența este împărțită în două subsecvențe egale și fiecare subsecvența este sortată. După sortare se interclasează cele două subsecvențe, astfel rezultă secvența sortată în întregime.

Pentru subsecvențe se aplică același abordare până când ajungem la o subsecvență elementară care se poate sorta fără împărțire (secvență cu un singur element).

# Python - Merge Sort

```
def mergeSort(l, start, end):  
    """  
        sort the element of the list  
        l - list of element  
        return the ordered list (l[0]<l[1]<...)  
    """  
    if end-start <= 1:  
        return  
    m = (end + start) // 2  
    mergeSort(l, start, m)  
    mergeSort(l, m, end)  
    merge(l, start, end, m)
```

Apel: mergeSort(l,0,len(l))

# Interclasare (Merging)

*Date*  $m, (x_i, i=1,m), n, (y_i, i=1,n);$

*Precondiții:*  $\{x_1 \leq x_2 \dots \leq x_m\}$  și  $\{y_1 \leq y_2 \leq \dots \leq y_n\}$

*Rezultate*  $k, (z_i, i=1,k);$

*Post-condiții:*  $\{k=m+n\}$  și  $\{z_1 \leq z_2 \leq \dots \leq z_k\}$  și  $(z_1, z_2, \dots, z_k)$  este o permutare a valorilor  $(x_1, \dots, x_m, y_1, \dots, y_n)$

complexitate interclasare:  $\theta(m+n)$ .

## Complexitate sortare prin interclasare

Caz favorabil = Caz defavorabil = Caz mediu

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \theta(n)$$

Complexitate:  $\theta(n \cdot \log n)$

Spațiu de memorare adițională pentru merge sort  $\theta(n)$

## **Curs 11: Sortări**

- **Algoritmi de sortare**
  - metoda bulelor, quick-sort, merge sort
- **Sortare in Python: sort, sorted**
  - parametrii: poziționali, prin nume, implicita, variabil
  - list comprehension, funcții lambda

## **Curs 12 – Technici de programare**

- ✦ **Divide-et-impera (divide and conquer)**
- ✦ **Backtracking**