# SEMINAR 11 – 111

December 13, 2021

1. Complexitati
2. Recursivitate
3. Sortari

## Complexitati

* Complexitate dpdv al timpului de executie
* Complexitate dpdv al memoriei folosite

n -> ∞

T(n) – numarul de pasi executati, numaram operatiile mai importante

Θ(n2) – complexitate exacta, tot timpul complexitatea ramane la fel

O(n2) – marginea superioara, de ex: 1, n, n2

Ω(n2) – marginea inferioara, de ex: n2, n3, n4….

De ex: T(n) = 3\*n3 + 20\*n + 137 ϵ O(n3)

ϵ Θ(n3)

Complexitatiile se analizeaza prin 3 cazuri:

1. Caz favorabil
2. Caz defavorabil
3. Caz mediu (media aritmetica a complexitatilor)

**def** cautare(l, el):  
 **for** e **in** l:  
 **if** e == el:  
 **return True  
 return False**

Caz favorabil: el == l[0] -> T(n) = 1 ϵ O(1)

Caz defavorabil: el == l[n-1] sau el not ϵ l -> T(n) = n ϵ Θ(n)

Caz mediu:

T(n) = (1+2+ …+n)/n = ( n+1 ) /2 ϵ Θ(n)

Complexitatea finala: O(n)

## Recursivitate

**def** binary\_search(l, low, high, el):  
 *'''  
 index la care se afla el daca e in lista  
 altfel, -1* **:param** *l: l ordonata crescator* **:param** *el:* **:return***:  
 '''* **if** high>=low:  
 middle = (high + low) //2  
 **if** el == l[middle]:  
 **return** middle  
 **elif** l[middle] > el:  
 **return** binary\_search(l, low, middle - 1, el)  
 **else**:  
 **return** binary\_search(l, middle + 1, high, el)  
 **else**:  
 **return** -1

T(n) = { 0, lista este vida, n=0

3 + T(n/2), n>=1

Notam n = 2k 🡺 k = log2n

T(2k) = 3 + ~~T (2~~~~k-1~~~~)~~

~~T (2~~~~k-1~~~~)~~ = 3 + ~~T (2~~~~k-2~~~~)~~

….

~~T(2)~~ = 3 + ~~T(1)~~

~~T(1)~~ = 3 + T(0)

------------------- +

T(2k) = k \* 3 + T(0) = 3\*k +1 = 3 \* log2n + 1 ϵ O(log2n)

## Sortari

### Selection Sort (2 for-uri imbricate)

**def** selection\_sort(arr):  
 **for** poz **in** range(0, len(arr) - 1):  
 pozMin = poz  
 **for** i **in** range(poz + 1, len(arr)):  
 **if** arr[i] < arr[pozMin]:  
 pozMin = i  
 **if** pozMin > poz:  
 arr[pozMin], arr[poz] = arr[poz], arr[pozMin]  
 **return** arr

Caz fav = Caz mediu = Caz nefavorabil

Θ(n2)

### Insertion sort (for + while)

[-------------]el [------------------]

Sortat nesortat

El -> inserat in partea de sir sortat a.i. sa se pastreze ordinea de sortare

**def** insertion\_sort(arr):  
 **for** i **in** range(0, len(arr)):  
 ind = i - 1  
 valCurenta = arr[i]  
 **while** ind >= 0 **and** valCurenta < arr[ind]:  
 arr[ind + 1] = arr[ind]  
 ind-=1  
 arr[ind + 1] = valCurenta  
 **return** arr

Caz favorabil: arr sortat crescator T(n) = n ϵ Θ(n)

Caz defavorabil: arr sortat descrescator T(n) = n2 ϵ Θ(n2)

Caz mediu: ….. … T(n) ϵ Θ(n2)

Complexitatea: O(n2)

### Merge Sort

**def** interclasare(l, st1, dr1, st2, dr2):  
 rez = []  
 **while** st1<dr1 **and** st2<dr2:  
 **if** l[st1] < l[st2]:  
 rez.append(l[st1])  
 st1+=1  
 **else**:  
 rez.append(l[st2])  
 st2+=1  
 **while** st1< dr1:  
 rez.append(l[st1])  
 st1+=1  
 **while** st2< dr2:  
 rez.append(l[st2])  
 st2+=1  
 **return** rez  
  
**def** mergeSortRec(l, start, end):  
 **if** end - start > 1:  
 m = (start + end) // 2  
 mergeSortRec(l, start, m)  
 mergeSortRec(l, m, end)  
 lista = interclasare(l, start, m, m, end)  
 **for** i **in** range(len(lista)):  
 l[i+start] = lista[i]  
 **return** l

Caz favorabil: nu este

T(n) = { 0, n<=1

2 \* T(n/2) + n (n- numarul de pasi de la interclasare), n> 1

Notam n = 2k 🡺 k = log2n

T(2k) = 2 \* T(2k-1) + 2k

T(2k-1) = 2 \* T(2k-2) + 2k-1 \ \*2

T(2k-2) = 2 \* T(2k-3) + 2k-2  \ \* 22

….

T(2) = 2 \* T(1) + 2 \ \* 2k-1

----------------------------------+

T(2k) = 2 \* T(2k-1) + 2k

2 T(2k-1) = 4 \* T(2k-2) + 2k \ \*2

22 T(2k-2) = 8 \* T(2k-3) + 2k  \ \* 22

….

2k-1 T(2) = 2k \* T(1) + 2k \ \* 2k-1

---------------------------------------- +

T(2k) = k \* 2k

T(n) = n log2n ϵ Θ (nlog2n)

Complexitatea dpdv al memorie: Θ (nlog2n)

### Quick Sort

Complexitate medie si caz favorabil: Θ(nlog2n)

Complexitate caz defavorabil: Θ(n2)