ANALIZA COMPLEXITĂȚII ALGORITMILOR RECURSIVI

ALG. RECURSIVI sunt ușor de implementat → execuție cu costuri suplimentare → apelurile recursive necesită mem. suplim. (STIVA PROGRAMULUI).

Ex: function fact(n)

```
function fact(n)
    if (n<=1) return 1
    else return n*fact(n-1)
```

n=4 ⇒ fact(4) [ ]

↳ 4·fact(3) [4]

↳ 3·fact(2) [3,4]

↳ 2·fact(1) [2,3,4]

↳ 1 [1,2,3,4]

ETAPELE ANALIZEI ALG. ITERATIVI	ET. ANALIZEI ALG. REC
1. STABILIREA DIM. DATELOR DE INTRARE	
2. IDENTIF. OP. DOMINANTE și NR. DE REPETĂRI AL AC. DACĂ AC NR. DEPINDE DE PROP. D.I.: CCMF, CCMD, (CM)	2. IDENTIFICAREA RECURSIEI PE ANUMITE CAZURI INIȚIALE și DEF. RELAȚIEI DE RECURSIE PT. OP. DOM.
3. DETER. EXPR. MATE a T.E. și ORDINUL DE COMPLEXITATE	
prin însumarea nr. de repetări ale instr. alg.	prin folosirea unor met. speciale (substituție, iterației, MASTER, etc.)

În cazul alg. recursivi nu putem

să știm dinainte câte apeluri recursive se vor efectua de aceea e nevoie să det. relația de recursie a fcției recursive analizate și în baza ac. relații (prin aplicarea metodelor specifice) să se det. expr. mate T.E.

În analiza complex. fcțiilor recursive, condiția de oprire nu se ia în calcul deoarece ac. are de regulă un cost unitar.

METODA SUBSTITUȚIEI: se aplică relației de recursie.

- met. subst. înainte
- met. subst. înapoi.

1. METODA SUBSTITUȚIEI ÎNAINTE: pornește de la un pas inițial și pe baza ac. se det. valoarea pt. pasul urm. Ac. procedeu continuă până când reușim să det. expr. mate. T.E. Nu se poate aplica (∀) relației de recursie.

costul nerecursiv.

Ex: Pp. $t(n)=t(n-1)+n$. (RELAȚIE DE RECURSIE)

De. condiția de oprire ($n=1$) $t(1)=1$ // cost constant.

$t(1)=1$.

$t(2)=t(1)+2=1+2$.

$t(3)=t(2)+3=1+2+3$.

$t(k)=t(k-1)+k=\overbrace{1+\ldots+k-1}^{t(k-1)}+k$

$\Rightarrow$ generalizăm în n:

$t(n)=1+2+\ldots+n=$

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

$t(n)=O(n^2)$.

```
funct(n)
  - - - - -      O(1)
  funct(n-1)
  - - - - -      O(1).
  for i ← 1,n
    // O(1)
```

2. METODA SUBSTITUȚIEI ÎNAPOI

Relația de recursie ⇒ expr. note T.E. (ca și toate celelalte metode studiate)

În ac. noua val. pasului curent coresp. relației de recursie sunt substituite cu val. corespunzătoare pasului anterior. Substituțiile se repetă până când este atins pasul inițial

Ex: $t(n) = t(n-1) + n$ - - - - - - - - $\Rightarrow t(1)$

MET. SUBST. ÎNAPOI

$$t(n) = t(n-1) + n = [\underbrace{t(n-2) + n-1}_{t(n-1)}] + n = t(n-2) + (n-1) + n = [\underbrace{t(n-3) + (n-2)}_{t(n-2)}] + (n-1) + n = t(n-3) + (n-2) + (n-1) + n = \dots = t(n-k) + (n-(k-1)) + \dots + (n-1) + n =$$

$$= t(n-k) + (n-\underset{\substack{\uparrow \\ (n-1)}}{k}+1) + \dots + (n-1) + n \overset{}{\underset{k=n-1}{=\!=}} t(1) + (n-(n-1)+1) + \dots + (n-1) + n$$

$$= \boxed{t(1)} + 2 + \dots + (n-1) + n = \boxed{1} + 2 + \dots + (n-1) + n = \frac{n(n+1)}{2} \Rightarrow$$

$$t(n) = O(n^2)$$

APLICAȚIE PT. METODA SUBSTITUȚI

ALGORITMUL DE CĂUTARE BINARĂ.

```
function binSearch(start, finish)
    if (start = finish)
        if (v[start] = x) return true
        else return false
        □
    else
        mij ← (start + finish)/2
        if (x ≤ v[mij])
            binSearch(start, mij)
        else binSearch(mij+1, finish)
        □
    □
□
```

PROP. D.I. $v[1 \dots n]$ - sortat crescător

APELUL INIȚIAL: binSearch(1, n).

$t(n) = t(n/2) + 1.$

$\underline{t(1)} = \underline{1}.$

Met. subst. înapoi:

$t(n) = t(n/2) + 1 = [\underbrace{t(n/2^2) + 1}_{t(n/2)}] + 1$

$= t(n/2^2) + 2 = [\underbrace{t(n/2^3) + 1}_{t(n/2^2)}] + 2$

$= t(n/2^3) + 3 = \dots$

$= t(\underbrace{n/2^k}_{1}) + k =$

1 (ca să aj. cond. oprire)

$\frac{n}{2^k} = 1 \Rightarrow k = \log_2 n$

$\overset{}{\underset{k=\log_2 n}{=\!=\!=}} t(n/n) + \log_2 n =$

$= t(1) + \log_2 n =$

$= 1 + \log_2 n$

$\Rightarrow t(n) = O(\log n)$

METODA ITERAȚIEI

Transf. recursia într-o sumă folosind teh. de mărginire a sumelor. Conversia recursiei în sumă se face prin expandarea recursiei și expr. ei ca o sumă de termeni dependenți de n și de condițiile inițiale.

EX: $t(n) = t(n-1) + 1$, $t(0) = 1$.

suma(n)

```
if (n=0) return 0
else return n + suma(n-1)
```

$$t(n) = \cancel{t(n-1)} + 1 \quad (k=1)$$

$$\cancel{t(n-1)} = \cancel{t(n-2)} + 1 \quad (k=2)$$

$$\cancel{t(n-2)} = \cancel{t(n-3)} + 1$$

$$\ldots$$

$$\cancel{t(n-(k-1))} = \cancel{t(n-k)} + 1$$

$k=n$ … $k=n$

$$\cancel{t(1)} = t(0) + 1. \quad (k=n)$$

$k=n$

$+$

$$t(n) = t(0) + \underbrace{1 + \ldots + 1}_{n \text{ relații}} = 1 + n \cdot 1 = n + 1 \Rightarrow t(n) = O(n) \text{ - complex. liniară.}$$

ALGORITMUL MERGESORT

```
function mergeSort(start, finish)
    if (start < finish)
        mid ← (start + finish) / 2
        mergeSort (start, mid)
        mergeSort (mid + 1, finish)
        merge(start, mid, finish)
    □
□

merge(start, mid, finish) → O(n)
    for i ← start, finish          } O(n)
        temp[i] ← vect[i]
    □
    i ← start, j ← mid + 1, k ← start
    while (i <= mid && j <= finish) --ecMMD--> O(n/2)
        if (temp[i] ≤ temp[j])
            vect[k++] ← temp[i++]
        else
            vect[k++] ← temp[j++]
        □
    □
    while (i <= mid) --ecMMD--> O(n/2)
        vect[k++] ← temp[i++]
    □
    while (j <= finish) --ecMMD--> O(n/2)
        vect[k++] ← temp[j++]
    □
□

mergeSort(1, n).
```

$$\boxed{t(n) = 2 \cdot t(n/2) + n}$$