Recapitulare (formule matematice utile in analiza algoritmilor)

$$\sum_{i=1}^{N} i = \frac{N(N+1)}{2} \approx \frac{N^2}{2}$$

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6} \approx \frac{N^3}{3}$$

$$\sum_{i=1}^{N} i^k = \frac{N^{k+1}}{|k+1|} \qquad k \neq -1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1}-1$$

$$\sum_{i=0}^{n} a^i \leq \frac{1}{1-a} \qquad \text{if } 0 < a < 1$$

$$\sum_{i=0}^{n} a^i \to \frac{1}{1-a} \qquad \text{if } 0 < a < 1 \text{ and } n \to \infty$$

$$\sum_{i=1}^{N} f(N) = Nf(N)$$

$$\sum_{i=n_0}^{N} f(i) = \sum_{i=1}^{N} f(i) - \sum_{i=1}^{n_0-1} f(i)$$

$$\log_a b = \frac{\log_c b}{\log_c a} \quad ; c > 0$$

$$\log ab = \log a + \log b$$

$$\log a / b = \log a - \log b$$

$$\log(a^b) = b \log a$$

$$\log x < x \quad \text{for all } x > 0$$

Recurente de complexitate – exercitii *(a se rezolva de catre studenti, iesind la tabla)*

1. T(n) = T(n-1) + n
2. T(n) = T(n/2) + log n
   - se discuta metoda master, ar fi cazul 3 dar nu se respecta conditia
   - se rez prin metoda iterativa
   - se introduce metoda schimbarii de variabila, n=2^k => T(2^k)=T(2^(k-1))+k=> =>G(k)=G(k-1)+k (rec de la pct 1) => G(k)=theta(k^2) => T(n)=theta(log^2 n)
3. T(n) = T(sqrt(n)) + 1
   - hint: folositi schimbarea de variabila
   - se rezolva similar, cu n=2^k, G(k)=G(k/2)+1, met master caz 2 => log log n
4. T(n) = T(n/2) + 2T(n/4) + cn
   - se cere sa se rezolve fol arb de recurenta
   - arborele genereaza cai mai lungi si cai mai scurte, lim inf complexitate = cn*nr de nivele pana se termina cea mai rapida cale, lim sup = ...cm lenta, se observa ca ambele sunt de ord n log n si apoi se dem ghiceala prin met substitutiei
5. Lucrare: T(n) = 3T(n/4) + n (la cealalta grupa 5T(n/6) + theta(n), dar e practic totuna) => se obtine o serie geometrica a unui nr subunitar care va tinde la o constanta, deci theta(n); dupa lucrare le-am aratat ca isi puteau confirma sau infirma rezultatul fol met master

Complexitate spatiala
- spre deosebire de timp, spatiul e refolosibil
- ce ne intereseaza este maximul de memorie pe care algoritmul il ocupa la un moment dat
- algoritmii recursivi tin minte structurile de date si, in plus, stiva de apeluri recursive
- memoria ocupata de aceasta stiva este limitata superior de dimensiunea maxima ocupata de contextul unui apel * adancimea in recursivitate

Studiu de caz: mergesort
- memoria ocupata de structurile de date: O(n)
- pt stiva de apeluri recursive: O(log n)
- in total O(n)

- totusi, in practica, e important daca e n sau 2n
(se deseneaza cum lucra mergesort anterior si cum lucreaza mergesort cu imbunatatirea de mai jos)

Mergesort cu imbunatatiri ale memoriei ocupate (fata de varianta din laboratorul anterior)

```
merge(start, mijloc, stop)
{
 for i=start to mijloc
   b[i]=a[i] // copiaza prima jumatate sortata intr-un vector auxiliar b

 i=start; j=mijloc+1; k=start;

 // copiaza inapoi in a pe cel mai mic dintre elementele curente in
 // cele 2 jumatati
 while (k<j) and (j<=stop)
   if b[i]<=a[j] then
     a[k++]=b[i++]
   else
     a[k++]=a[j++]

 // copiaza ce a mai ramas din prima jumatate, daca in ea a mai ramas
 while k<j
   a[k++]=b[i++]
 // pt a doua nu mai e nevoie, elementele sunt deja la locul lor!
}
```