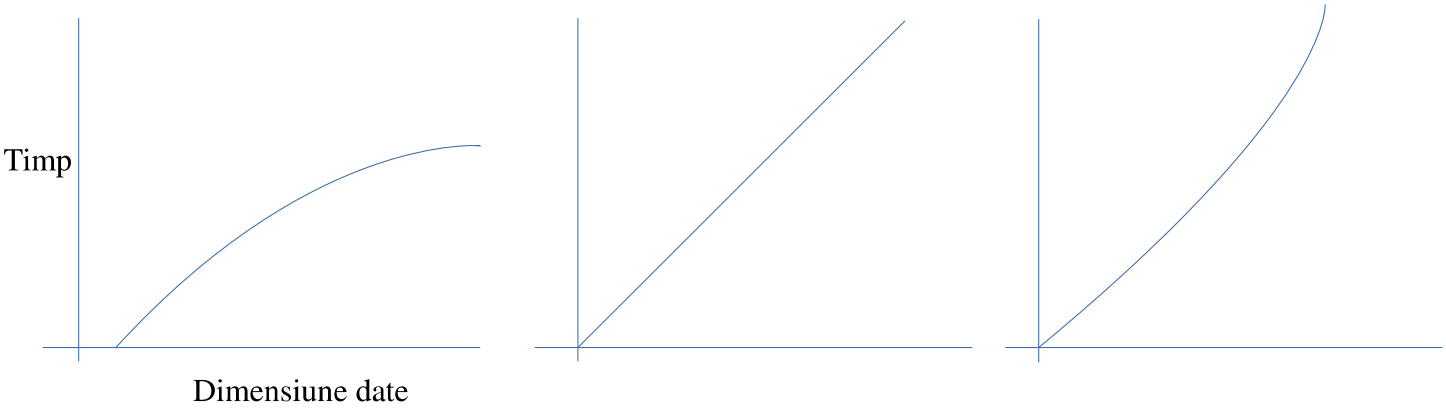
Algoritmi si Structuri de Date

- seminar I -

Complexitate.

Suntem interesati de complexitatea unui algoritm pentru a sti cum se comporta acesta pe masura ce dimensiunea datelor creste. Cu cat timpul de executie este mai mic, cu atat algoritmul este mai eficient.



Notatie asimptotica.

Vom masura complexitatea unui algoritm in functie de numarul instructiunilor simple (adunari, scaderi, asignari, etc.) pe care acesta le va executa. Vom considera ca toate intructiunile simple au acelasi timp de executie (si anume constant = O(1)).

Clase de complexitati.

- Ω - "best case": in cel mai bun caz, algoritmul are acea complexitate
- O - "worst case": in cel mai rau caz, algoritmul are acea complexitate
- Θ - "average": in general, algoritmul are acea complexitate

Definitii.

- $f(n) = \Omega(g(n)) <==> \exists\ c > 0, \exists\ n_0 \in N$ a.i. $f(n) >= c * g(n)\ \forall\ n >= n_0$
- $f(n) = O(g(n)) <==> \exists\ c > 0, \exists\ n_0 \in N$ a.i. $f(n) <= c * g(n)\ \forall\ n >= n_0$
- $f(n) = \Theta(g(n)) <==> \exists\ c_1 > 0, \exists\ c_2 > 0, \exists\ n_0 \in N$ a.i.

$$c_1 * g(n) <= f(n) <= c_2 * g(n)\ \forall\ n >= n_0$$

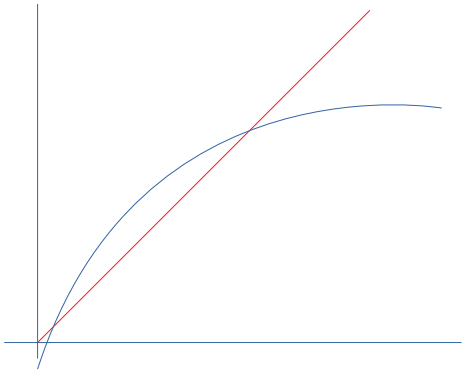
Exemple:

- f(n) = 2 * n + 5
  - f(n) = $\Omega(1)$ (pentru ca $\exists$ c = 1, $\exists$ n_0 = 3 a.i. f(n) >= 1 $\forall$ n >= n_0)
    - Desi orice algoritm este in $\Omega(1)$, acest lucru nu spune nimc despre algoritm, am vrea sa stim cea mai mare functie g a.i. f(n) = Ω(g(n)), in acest caz ar fi g(n) = n.
  - f(n) = $O(n^2)$ (pentru ca $\exists$ c = 1, $\exists$ n_0 = 4 a.i. f(n) <= n^2 $\forall$ n >= n_0)
    - Acest lucru spune ca functia f nu creste mai repede decat functia n^2, dar exista de asemenea si o complexitate mai buna pe care putem sa o gasim: g(n) = n (pentru ca $\exists$ c = 5, $\exists$ n_0 = 2 a.i. f(n) <= 5 * n $\forall$ n >= n_0).
    - Pentru a gasi acea constanta puteti inlocui

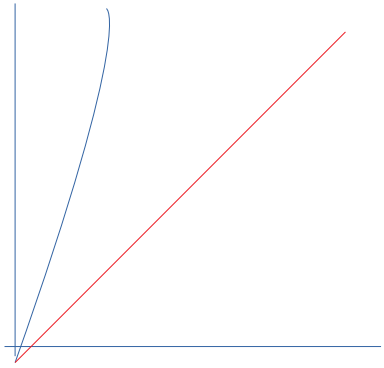
      f(n) <= c * n => 2 * n + 5 <= c * n => c >= 2 + 5/n => c = 3;
      2 * n + 5 <= 3 * n =>5 <= n; n_0 = 5;
      pentru c = 3 si n_0 = 5, 2 * n + 5 <= 3 * n $\forall$ n >= 5
      (verificare: n = 5, 2 * 5 + 5 <= 3 * 5 – ok, n = 6, 2 * 6 + 5 <= 3 * 6 – ok)
  - f(n) = Θ(n) (pentru ca $\exists$ c_1 = 1, $\exists$ c_2 = 3, $\exists$ n_0 = 5 a.i.

    g(n) <= f(n) <= 3 * g(n) $\forall$ n >= 5)
    - Nu putem gasi intotdeauna o functie g pentru a spune ca un algoritm este in Θ(g(n)), dar de obicei, daca putem gasi ca f(n) = Ω(g(n)) si f(n) = O(g(n)), atunci putem spune ca f(n) = Θ(g(n)).

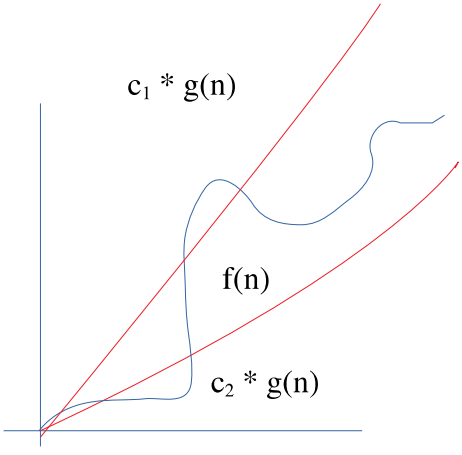
Vizual.



log(n) = O(n)



2^n = Ω(n)



f(n) = Θ(g(n))

Exercitii.

1. Spuneti in ce clasa se afla f in raport cu g ({f = Ω(g)} sau {f = O(g)}, sau {f = Ω(g) si f = O(g) deci f = Θ(g)})

(a) $f(n) = 2 * n + 5$ $g(n) = 2^n + 3 * n$

(b) $f(n) = sqrt(n) * 1000$ $g(n) = n + 1$

(c) $f(n) = n * sqrt(n)$ $g(n) = n^2$

(d) $f(n) = n$ $g(n) = \log^3(n)$

(e) $f(n) = n!$ $g(n) = 2^n$

(f) $f(n) = 2^n$ $g(n) = 3^n$

(g) $f(n) = \log_3(n)$ $g(n) = \log_5(n)$

(puteti rezolva usor unele subpuncte folosind lim (n→inf) f(n)/g(n))

Structuri de date abstracte.

Sunt stucturi care pot tine date arbitrare (numere, stringuri, caiete, etc.).

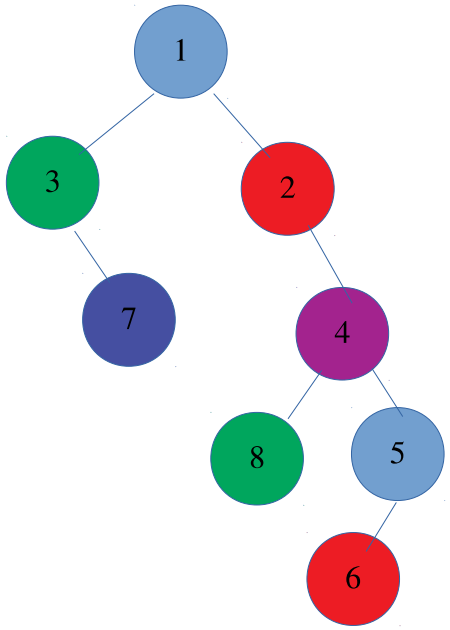
Structuri de date si operatii:

- Liste simplu si dublu inlantuite
  - adaugare element (inceput, mijloc, final)
  - stergere element (inceput, mijloc, final)
  - cautare element in lista
- Stive
  - adaugare la final
  - extragere de la final
- Cozi
  - adaugare la final
  - extragere de la inceput
- Arbori binari de cautare
  - adaugare element in arbore
    - se porneste cu nodul curent ca fiind radacina
    - daca nodul curent este NULL, se insereaza pe pozitia curenta

        - altfel, se compara elementul care trebuie inserat cu valoarea nodului curent
            - daca este mai mic, se initializeaza nodul curent ca fiind fiul stang
            - daca este mai mare, se initializeaza nodul curent ca fiind fiul drept
    - cautare element in arbore
        - se porneste cu nodul curent ca fiind radacina
        - daca nodul curent este elementul cautat, se returneaza nodul respectiv
        - altfel, se compara elementul care trebuie inserat cu valoarea nodului curent
            - daca este mai mic, se initializeaza nodul curent ca fiind fiul stang
            - daca este mai mare, se initializeaza nodul curent ca fiind fiul drept
    - parcurgeri (inordine - SRD, preordine - RSD, postordine- SDR)
        - se porneste cu nodul curent ca fiind radacina
        - se urmaresc literele parcurgerii, astfel:
            - S – stanga => nodul curent devine fiul stang
            - D – dreapta => nodul curent devine fiul drept
            - R – radacina => se afiseaza valoarea nodului curent
    - stergere element din arbore
        - se cauta elementul respectiv in arbore
        - daca nodul are doar un fiu, se sterge direct
        - altfel, se inlocuieste cu predecesorul/succesorul sau din parcurgerea SRD
        - se continua recursiv stergerea
    - reconstructie arbore avand parcurgerile SRD si RSD
        - se incepe cu parcurgerea RSD (de aici vom lua valorile una cate una)
        - primul element din parcurgerea RSD este radacina arborelui
        - cautam in SRD pozitia acestuia si aflam ce elemente se afla in stanga si ce elemente se afla in dreapta radacinii
        - continuam cu parcurgerea RSD si aflam care este urmatorul nod din arbore

Exemplu reconstructie arbore avand parcurgerile SRD si RSD:

- SRD: 3 7 1 2 8 4 6 5
- RSD: 1 3 7 2 4 8 5 6



(1) Radacina este primul element din RSD. **1** 3 7 2 4 8 5 6

(2) Cautam valoarea radacinii in SRD: 3 7 **1** 2 8 4 6 5.
Elementele 3 si 7 se afla in partea stanga.
Elementele 2, 8, 4, 6, 5 se afla in partea dreapta.

(3) Continuam cu urmatorul element din RSD ~~1~~ **3** 7 2 4 8 5 6

(4) Cautam valoarea 3 in SRD: **3** 7
In stanga lui 3 nu avem niciun element.
In dreapta lui 3 se afla elementul 7.

(5) Urmatorul element din RSD este ~~1 3~~ **7** 2 4 8 5 6

(6) Cautam valoarea 7 in SRD: 7
In stanga lui 7 nu se afla nimic
In dreapta lui 7 nu se afla nimic (e frunza)

(7) Am terminat partea stanga (cea cu verde de la pasul 2)
Continuam cu partea dreapta.

(8) Urmatorul element din RSD este ~~1 3 7~~ **2** 4 8 5 6

(9) Cautam elementul 2 in SRD: **2** 8 4 6 5
In stanga lui 2 nu se afla nimic
In dreapta lui 2 se afla elementele 8 4 6 5

(10) Urmatorul element din RSD este ~~1 3 7 2~~ **4** 8 5 6

(11) Cautam elementul 4 in SRD: 8 4 6 5
In stanga lui 4 se afla 8
In dreapta lui 4 se afla elementele 6 5

(12) Urmatorul element din RSD este ~~1 3 7 2 4~~ **8** 5 6

(13) Cautam elementul 8 in SRD (pasul 11): 8 – este frunza

(14) Urmatorul element din RSD este ~~1 3 7 2 4 8~~ **5** 6

(15) Cautam elementul 5 in SRD (pasul 11): 6 5
In stanga lui 5 se afla elementul 6
In dreapta lui 5 nu se afla niciun element

(16) Urmatorul element din RSD este ~~1 3 7 2 4 8 5~~ **6**
Elementul 6 se afla in stanga lui 5 (pasul 15).

Complexitati operatii.

	Adaugare element	Cautare element	Stergere element
Lista simplu inlantuita (implementare – alocare dinamica)	Inceput O(1) Sfarsit O(n) (sau eficient O(1)) Mijloc O(k) (sau cu poz. O(1))	O(n)	O(n) (sau cu poz. O(1))
Lista simplu inlantuita (implementare – vector)	Inceput O(n) Sfarsit O(1) Mijloc O(k)	O(n)	O(n) (sau eficient O(1))
Stiva	O(1)	O(n)	O(1)
Coada	O(1)	O(n)	O(1)
Arbore binar de cautare	O(h)	O(h)	O(h)

Exercitii.

1. Inserati urmatoarele valori, pe rand, intr-un arbore binar de cautare: 7 5 6 8 10 9 1. Stergeti elementul 7.

2. Pot urmatoarele elemente fi considerate secvente de cautare in arbore binar de cautare:
   - 30 11 15 14 10 3 1
   - 100 50 25 13 17 14 15

3. Reconstruiti arborele cu parcurgerile:

   RSD: 1 5 8 3 7 4 12 — RSD: 7 9 10 4 3 1 6 8 5

   SRD: 5 3 8 1 7 12 4 — SRD: 10 4 9 3 7 6 8 1 5

4. Aveti la dispozitie trei structuri de date (1 → stiva, 2 → coada, 3 → stiva). Puteti aplica urmatoarele operatii:
   - caracter → introducere caracterul in prima structura de date
   - operatie de tip 1 → scoatem din prima structura, introducem in a doua
   - operatie de tip 2 → scoatem din a doua structura, introducem in a treia
   - operatie de tip 3 → scoatem din a treia structura, introducem in prima

Scrieti un sir de operatii, astfel incat, la finalul executiei lor, sa obtinem:
- in prima structura cuvantul: Afara
- in a doua structura cuvantul: e
- in a treia strunctura cuvantul: soare

Probleme intersante.

1. Se da o lista simplu inlantuita. Stergeti al k-lea element de la final (cu O(1) memorie).
2. Se da o lista simplu inlantuita. Inversati lista (complexitate O(n) timp, O(1) memorie).
3. Se da o lista simplu inlantuita. Gasiti un algoritm de complexitate liniara care afiseaza daca acea lista este palindromica.
4. Implementati o structura de date care poate raspunde rapid la urmatoarele query-uri:
    - inserare element la final
    - stergere element de la final
    - afisare element maxim
5. Simulati o coada folosind doua stive.
6. Simulati o stiva folosind doua cozi.
7. Primiti un arbore binar. Gasiti un algoritm care verifica daca este arbore binar **de cautare.**
8. Cautare binara: infoarena: loto, fact, transport.